Karlsruhe Institute of Technology

Fault-Tolerance and Deaggregation Security of Aggregate Signatures

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Björn Kaidel

aus Eberbach

Tag der mündlichen Prüfung: 09.12.2019

Erster Referent: Prof. Dr. Jörn Müller-Quade
Zweiter Referent: Prof. Dr. Tibor Jager

Acknowledgments

I would like to thank Jörn Müller-Quade, for giving me the opportunity to work at his institute and for supervising my thesis, and Tibor Jager, for co-refereeing my thesis. I also want to thank them for their interesting lectures during my time as a student that have strongly influenced my decision to work in cryptography.

Furthermore, I would like to thank my co-authors, namely Alexander Koch, Jessica Koch, Gunnar Hartung and Andy Rupp, for their motivation and many inspiring research discussions. It was a pleasure to work with you!

I would like to extend further thanks to all colleagues that I've had the pleasure of working with during my time at the institute. In particular, I want to express my gratitude to my friend Brandon Broadnax for his invaluable insights and for supporting me in many ways.

Many thanks also go to Thomas Agrikola, for motivating talks; Matthias Gabel, for interesting (musical) discussions; to Willi Geiselmann, for our work together in teaching, support and for great lectures during my time as a student; to Dennis Hofheinz, for many fantastic lectures and motivating me to work in cryptography; to Alexander Koch, for our work on the lecture "Sicherheit" and interesting conversations; to Gunnar Hartung, for our work on the lecture "Digitale Signaturen" and many research discussions; to Julia Hesse, for asking an important question that started the fault-tolerance research group; to Jessica Koch for motivating discussions; to Lisa Kohl, for many interesting conversations and support and to Bernhard Löwe for lending me his ear more than once.

I would also like to thank my friends Micha Schäfer and Gunter Kopf for the right words at the right time and for offering new perspectives. Furthermore, big thanks go to my friends Josefina Paulson, Claudia Richter, Rick Krüger and Sebastian Elsner for our musical adventures.

Last, but most certainly not least, I would like to extend my deepest gratitude to my wife Regina Kunkel, as well as her family, and my parents Jurita and Gerd Kaidel, for their unwavering love and support. Without them, this thesis would not have been possible.

Zusammenfassung

Ein zentrales Problem der digitalen Kommunikation ist die Absicherung der *Authentizität* und *Integrität* digitaler Dokumente, wie etwa Webseiten, E-Mails oder Programmen. So soll beispielsweise für den Empfänger einer E-Mail nachvollziehbar sein, dass die empfangene E-Mail tatsächlich vom angegebenen Absender stammt (Authentizität) und nicht durch Dritte verändert wurde (Integrität). Digitale Signaturen sind ein Hauptwerkzeug der Kryptographie und IT-Sicherheit, um diese Eigenschaften zu gewährleisten.

Hierzu wird vom Absender ein geheimer Schlüssel verwendet, um für das zu sichernde Dokument eine *Signatur* zu erstellen, die mithilfe eines öffentlich bekannten Verifikationsschlüssels jederzeit überprüft werden kann. Die Sicherheitseigenschaften solcher digitaler Signaturverfahren garantieren sowohl, dass jede Änderung am Dokument dazu führt, dass diese Überprüfung fehlschlägt, als auch dass eine Fälschung einer Signatur praktisch unmöglich ist, d.h. ohne den geheimen Schlüssel kann keine gültige Signatur berechnet werden. Somit kann bei einer erfolgreichen Verifikation davon ausgegangen werden, dass das Dokument tatsächlich vom angegebenen Absender erstellt und seit der Berechnung der Signatur nicht verändert wurde, da nur der Absender über den geheimen Schlüssel verfügt.

Aggregierbare Signaturen bieten zusätzlich die Möglichkeit Signaturen mehrerer Dokumente zu einer einzigen Signatur zusammenzuführen bzw. zu *aggregieren*. Diese Aggregation ist dabei jederzeit möglich. Eine aggregierte Signatur bezeugt weiterhin sicher die Integrität und Authentizität aller ursprünglichen Dokumente, benötigt dabei aber nur so viel Speicherplatz wie eine einzelne Signatur. Außerdem ist die Verifikation einer solchen aggregierten Signatur üblichrweise schneller möglich als die sukzessive Überprüfung aller Einzelsignaturen. Somit kann die Verwendung eines aggregierbaren Signaturverfahrens anstelle eines gewöhnlichen Verfahrens zu erheblichen Verbesserungen der Performanz und des Speicherverbrauchs bei Anwendungen von Signaturen führen.

In dieser Dissertation werden zwei zusätzliche Eigenschaften von aggregierbaren Signaturverfahren namens *Fehlertoleranz* und *Deaggregationssicherheit* untersucht. Fehlertoleranz bietet eine Absicherung des Verfahrens gegen fehlerhafte Signier- und Aggregationsvorgänge und Deaggregationssicherheit schützt vor ungewollten Löschungen. Beide Eigenschaften werden im Folgenden erläutert.

Fehlertoleranz: Durch System- und Programmfehler, sowie inkorrektes oder auch bösartiges Nutzerverhalten ist es möglich, dass fehlerhafte Einzelsignaturen zu einer bestehenden aggregierten Signatur hinzugefügt werden. Alle bisherige aggregierbaren Signaturverfahren haben jedoch den Nachteil, dass bereits das Aggregieren einer einzigen fehlerhaften Einzelsignatur dazu führt, dass auch die

aggregierte Signatur fehlerhaft und somit unbrauchbar wird. Die aggregierte Signatur kann danach nicht mehr korrekt verifiziert werden. Insbesondere kann aus ihr nun keinerlei Aussage mehr über die Integrität und Authentizität der Dokumente abgeleitet werden, die vor dem Hinzufügen der fehlerhaften Einzelsignatur korrekt signiert wurden. Dies hat zur Folge, dass alle gegebenen Sicherheitsgarantien verloren gehen und es wird ein aufwändiges Neusignieren aller Dokumente notwendig, welches unter Umständen und je nach Anwendung nur schwer bis überhaupt nicht möglich ist.

In dieser Dissertation wird das erste fehlertolerante aggregierbare Signaturverfahren vorgestellt, bei dem das Hinzufügen einzelner falscher Signaturen bis zu einer gewissen Grenze keine schädlichen Auswirkungen hat. Eine aggregierte Signatur wird erst dann ungültig und unbrauchbar, sobald die Anzahl hinzugefügter fehlerhafter Signaturen diese Grenze überschreitet und behält davor weiterhin seine Gültigkeit für die korrekt signierten Dokumente. Dazu wird ein Verfahren vorgestellt, mit dem *jedes beliebige* aggregierbare Signaturverfahren in ein fehlertolerantes Verfahren transformiert werden kann. Das zugrundeliegende Verfahren wird dabei nur als *Black-Box* verwendet und der Schutz gegen Fälschungsangriffe übertragt sich beweisbar und ohne Einschränkung auf das neue fehlertolerante Verfahren. Des Weiteren wird als Anwendung von fehlertoleranten Verfahren gezeigt, wie aus ihnen ein sicheres Log-Verfahren konstruiert werden kann.

Deaggregationssicherheit: Erlangt ein Angreifer Zugriff auf eine aggregierte Signatur für einen bestimmten Datensatz, so sollte es ihm nicht möglich sein aus diesem Aggregat eine gültige Signatur für einen Teil der geschützten Dokumente abzuleiten, indem er einzelne Signaturen entfernt oder *deaggregiert*. Solche Angriffe können für viele Anwendungsfälle problematisch sein, da so Signaturen für Mengen von Dokumenten berechnet werden könnten, die nicht von den eigentlichen Erstellern beabsichtigt waren und nie von ihnen selbst signiert wurden. Wird ein aggregierbares Signaturverfahren etwa verwendet um eine Datenbank abzusichern, so sollte es Angreifern nicht möglich sein einzelne Einträge daraus zu entfernen.

In dieser Dissertation werden mehrere Deaggregationssicherheitsbegriffe entwickelt, vorgestellt und untersucht. Dazu wird eine Hierarchie von verschieden starken Sicherheitsbegriffen entwickelt und die Zusammenhänge zwischen den einzelnen Begriffen werden formal untersucht. Dabei wird auch gezeigt, dass der von aggregierbaren Signaturverfahren garantierte Schutz gegen Fälschungen keinerlei Sicherheit gegen Deaggregationsangriffe gewährleistet. Des Weiteren wird die Deaggregationssicherheit einer Reihe von bekannten und wichtigen aggregierbaren Signaturverfahren näher betrachtet. Die von diesen Verfahren gebotene Sicherheit wird exakt klassifiziert, indem entweder Angriffsmöglichkeiten demonstriert werden oder formal bewiesen wird, welcher Sicherheitsbegriff der Hierarchie vom Verfahren erfüllt wird.

Außerdem wird die Verbindung von Fehlertoleranz und Deaggregationssicherheit untersucht. Dabei stellt sich heraus, dass beide Begriffe nicht zueinander kompatibel sind, indem bewiesen wird, dass fehlertolerante aggregierbare Signaturverfahren keinerlei Sicherheit gegen Deaggregationsangriffe bieten können. Somit muss bei Anwendungen von aggregierbaren Verfahren genau abgewogen werden, welche der beiden Eigenschaften notwendig ist und ob zusätzliche Sicherheitsmaßnahmen angewendet werden müssen, um dieses Problem für die konkrete Anwendung zu beheben.

Abstract

A prominent problem of digital communication is the protection of the *authenticity* and *integrity* of digital documents, like websites, emails or programs. For example, the receiver of an email must be able to verify that the email was actually sent by the alleged sender (authenticity) and that it was not changed by a third party during transmission (integrity). Digital signature schemes are a central tool of cryptography and IT security used to achieve these properties.

For this, the sender uses a secret key to compute a *signature* for the document that is to be protected. This signature can then be verified at any time using a publicly known verification key. The security guarantees of such digital signature schemes ensure both that every change of the document results in a failed verification and that forging signatures is practically impossible, i.e. it is not possible to compute valid signatures without the secret key. This way, if the verification of a signature succeeds, one can be sure that the document was created by the alleged sender and was not modified since the signature was computed, because only the sender knows the secret key.

Aggregate signature schemes offer the additional functionality that existing signatures of multiple documents can be compressed or *aggregated* into one single signature. This aggregate signature still securely attests to the authenticity and integrity of all documents, but only uses as much memory space as one individual signature of a single document. Moreover, an aggregate signature can usually be verified faster than successively verifying each individual signature. Hence, using an aggregate signature scheme instead of a regular signature scheme can result in substantial improvements in the necessary memory space and performance of the application that uses the signature scheme.

In this thesis, two additional properties of aggregate signature schemes called *fault-tolerance* and *deaggregation security* are discussed. Fault-tolerance secures the scheme against incorrect signing and aggregation operations and deaggregation security offers protection against unwanted deletions. Both properties are explained in the following paragraphs.

Fault-Tolerance: Through system and program errors, as well as incorrect or malicious user behavior, it is possible that invalid or *faulty* signatures get added to an already aggregated signature. However, all known aggregate signature schemes have one disadvantage in common: the aggregation of a faulty signature renders the whole aggregate invalid and therefore unusable. After the faulty signature was added, the aggregate signature can no longer be correctly verified. In particular, no information whatsoever about the authenticity and integrity of the previously correctly signed documents can be inferred. As a consequence, all given security guarantees are lost and all documents need to be signed again,

which can be very costly or even impossible, depending on the circumstances and application.

In this thesis, we present the first fault-tolerant aggregate signature scheme for which the aggregation of faulty signatures up to a specific limit has no negative effect. An aggregate signature only then becomes invalid if the number of added faulty signatures exceeds this limit and otherwise stays valid for all previously correctly signed documents. We present a construction that can be used to transform *any* aggregate signature scheme into a fault-tolerant one. Here, the underlying scheme is only used as a *black-box* and we prove that the security against forgery attacks is transferred without restrictions to the resulting fault-tolerant scheme. Furthermore, as an application of fault-tolerant aggregate signatures, we show how a secure and robust logging scheme can be constructed from such a scheme.

Deaggregation Security: If an attacker gets a hold of an aggregate signature for a specific set of documents, then he should not be able to compute a valid signature for a subset of the documents by removing or *deaggregating* individual signatures. Such attacks can be problematic in many applications, since this way signatures could be created that validate sets of documents that were never intended by the document owners and were never properly and intentionally signed by them. For example, suppose that an aggregate signature scheme is used to protect a database. Here, an attacker should not be able to selectively remove individual entries.

In this thesis, we present and discuss several definitions of deaggregation security. We present a hierarchy of deaggregation security definitions of varying strength and discuss their relationships formally. We also show that the security against forgery attacks offered by aggregate signature schemes implies no protection whatsoever against deaggregation attacks. Furthermore, we investigate the deaggregation security of a number of known and important aggregate signature schemes. We exactly classify the security provided by them by either demonstrating attack possibilities or by formally proving which definition of the deaggregation security hierarchy is fulfilled by them.

Moreover, we discuss the connection between fault-tolerance and deaggregation security. As it turns out, fault-tolerance and deaggregation security are incompatible, which we show by proving that no fault-tolerant scheme can offer any form of deaggregation security. Therefore, if an aggregate signature scheme is used in an application, great care needs to be taken in deciding which of the two properties is necessary and if additional security measures need to be implemented to solve this problem for the specific application.

Contents

1 Introduction **1**
- 1.1 Background and Motivation . . . 1
  - 1.1.1 Fault-Tolerance . . . 3
  - 1.1.2 Deaggregation Security . . . 4
- 1.2 Contributions of this Thesis . . . 5
- 1.3 Structure of this Thesis . . . 7

2 Preliminaries **9**
- 2.1 Notation . . . 9
- 2.2 Basics of Cryptography . . . 10
  - 2.2.1 Computational Hardness Assumptions & Reductions . . . 10
  - 2.2.2 Digital Signatures . . . 12
  - 2.2.3 Hash Functions and the Random Oracle Model . . . 16
  - 2.2.4 Pairings . . . 17
- 2.3 Cover-Free Families . . . 19

3 Aggregate Signatures **21**
- 3.1 Claims and Claim Sequences . . . 21
- 3.2 Fully Flexible Aggregate Signatures . . . 23
  - 3.2.1 The BGLS Aggregate Signature Scheme . . . 25
  - 3.2.2 Overview of Known Fully Flexible Schemes . . . 26
- 3.3 Sequential Aggregate Signatures . . . 27
  - 3.3.1 Overview of Known Sequential Schemes . . . 30
- 3.4 Synchronized Aggregate Signatures . . . 30
  - 3.4.1 Overview of Known Synchronized Schemes . . . 33
- 3.5 Identity-Based Aggregate Signatures . . . 34

4 Fault-Tolerance **37**
- 4.1 Introduction . . . 37
  - 4.1.1 Contribution . . . 39
  - 4.1.2 Related Work . . . 40
  - 4.1.3 Overview . . . 40
- 4.2 Basic Idea of Our Construction . . . 40
- 4.3 Framework for Fault-Tolerance . . . 42
- 4.4 Fault-Tolerance and Signature Size . . . 47
- 4.5 A Construction from Cover-Free Families . . . 49
  - 4.5.1 Achieving Unbounded Aggregation . . . 55
  - 4.5.2 Additional Feature: Selective Verification . . . 56

4.5.3 Instantiation using a Cover-Free Family based on Polynomials over a Finite Field . . . 57
4.6 Fault-Tolerance for Sequential Schemes . . . 60
4.7 Application: Robust Secure Logging . . . 64
4.7.1 Introduction to Secure Logging . . . 64
4.7.2 Forward Security . . . 65
4.7.3 Formal Definition of Robust Secure Logging . . . 67
4.7.4 Construction of a Robust Secure Logging Scheme . . . 69

5 Deaggregation Security 71
5.1 Introduction . . . 71
5.1.1 Contribution . . . 72
5.1.2 Related Work . . . 73
5.1.3 Overview . . . 75
5.2 Definitions of Deaggregation Security . . . 75
5.2.1 The k-Element Aggregate Extraction Assumption . . . 76
5.2.2 nDEAGG Security . . . 77
5.2.3 nDEAGG$^+$ Security: Attacks using Additional Claims . . . 80
5.2.4 Deaggregation Security & Unforgeability . . . 82
5.2.5 Relationships between nDEAGG and nDEAGG$^+$ . . . 86
5.2.6 Discussion of nDEAGG and nDEAGG$^+$ Security . . . 96
5.2.7 ADEAGG Security: Security against Adaptive Attacks . . . 97
5.2.8 Adapting the Definitions to Sequential and Synchronized Aggregation . . . 105
5.2.9 Overview of the Deaggregation Security Definitions . . . 106
5.3 Deaggregation Security of Known Schemes . . . 107
5.3.1 The BGLS Aggregate Signature Scheme . . . 107
5.3.2 The BNN Aggregate Signature Scheme . . . 110
5.3.3 The SMD Aggregate Signature Scheme . . . 113
5.3.4 The LMRS Sequential Aggregate Signature Scheme . . . 114
5.3.5 The NEV Sequential Aggregate Data Scheme . . . 117
5.3.6 The LOSSW Sequential Aggregate Signature Scheme . . . 119
5.3.7 The AGH Synchronized Aggregate Signature Scheme . . . 126
5.3.8 The AGHRO Synchronized Aggregate Signature Scheme . . . 135
5.3.9 Overview of the Schemes . . . 138
5.4 Deaggregation Security & Fault-Tolerance . . . 139

6 Conclusion and Prospects 143

Bibliography 147

Author's Publications 159

Chapter 1

Introduction

1.1 Background and Motivation

Providing a secure way of ensuring *authenticity* and *integrity* of digital data and documents is a problem that has long been recognized as central in the cryptography and IT security community [DH76]. Given a digital document like an email, how can the receiver be sure that it was actually created or sent by the alleged creator (authenticity) and that it was not changed since its creation (integrity), for example by a third party during transmission?

These security properties are crucial for many types of data and applications. Consider, for example, the case of operating system updates. It must be enforced that the operating system only accepts legitimate updates of the developer of the system, otherwise attackers could easily install malicious code like viruses and key loggers through fraudulent updates. This could have disastrous effects, especially if the computer in question is used to control the security and safety of critical infrastructures like power plants, train stations, airports and so on. But also in more mundane scenarios, like online shopping and banking, the user wants to be sure that she is actually connected to the website of the store or bank in question and is not entering her private data on a fake website run by criminals.

To this end, Diffie and Hellman [DH76] introduced the concept of *digital signature schemes*. Like hand-written signatures in the analog world, these schemes are used to sign digital documents and data. A digital signature scheme provides three algorithms for key generation, signing and verifying. The key generation algorithm computes a key pair consisting of a secret key and a verification key. The secret key is used together with the signing algorithm to compute signatures for messages[1] and must be kept secret, as the name already suggests. The verification key is used to then verify such signatures using the verification algorithm. This algorithm will either reject or accept the signature for the given message by outputting "valid signature for this message" or "invalid signature for this message", respectively. The verification key can safely (and should be) made public to ensure that everyone is able to verify the validity of

[1] In the context of digital signatures, digital data and documents are usually referred to as "messages".

signatures, which is why it is also often simply called *public key*. The security properties of a digital signature scheme then guarantee the following:

1. It is practically impossible to compute signatures *without* the secret key, i.e. signatures cannot be forged.

2. The verification algorithm will reject any message-signature pair where the signature was not explicitly computed for this message.

3. Modifying either the message or its signature in any way will result in a rejection by the verification algorithm.

Now, given these properties, if one receives a message together with a digital signature *and* this signature withstands the verification process, i.e. the verification algorithm outputs "valid signature for this message", one can be sure that the message was actually signed by the alleged sender and was not changed since the signature was computed. Furthermore, since the verification key and the algorithms of the scheme are publicly available, everybody is able to verify signatures.

This way, digital signature schemes provide the sought after properties of integrity and authenticity. They have become an integral part of cryptography and are used in a myriad of applications (like, for example, secure communication on the internet or electronic payment systems) and as building blocks for more complex constructions and protocols ([Lys+99; Lys02; Gol04; Bon+07; Nak09; HJ12; Res18; CL19], to mention only a handful of publications in these areas).

However, using digital signature schemes also comes at a cost: For each protected document, an additional file, namely the signature, must be computed, sent and stored. While the size of a signature is typically far smaller then the signed data itself, if many documents need to be signed, this can drastically increase the necessary memory space and bandwidth use of the application. The performance is also negatively affected, since the signatures need to be created and regularly verified. Verification needs to be done more or less every time the data is accessed and is usually a computationally expensive operation.

In [Bon+03] Boneh, Gentry, Lynn and Shacham introduce a new type of digital signature scheme called *aggregate signature schemes* that seeks to alleviate some these disadvantages. Such a scheme provides an additional *aggregation algorithm* that can be used to compress a set of signatures into a single *aggregate signature*, even if the individual signatures were created by separate signers using different secret keys.

Aggregation is a public operation, so no secret information is necessary to aggregate, and additional signatures can always be added to an already aggregated signature. Considering signature size, an aggregate signature has the same size as an individual signature for a single message, no matter how many signatures are aggregated. Furthermore, the verification of an aggregate signature for a given set of messages is often faster than verifying individual signatures for all messages. For example, in the scheme of Boneh, Gentry, Lynn and Shacham, verifying an aggregate is *twice* as fast as individual verification. These properties are beneficial in a multitude of applications ranging from sensor networks, secure logging and secure routing to software authentication [Ken+00; Bon+03; MT08; AGH10; BGR14; Har+16]. Let us shortly outline how aggregate signature schemes can be used in some of these scenarios.

A well-known application of aggregate signatures are *sensor networks* [Bon+03; BNN07; AGH10]. Here, small sensors measure some aspect of their physical surroundings and send the measurements to a central base station for processing. Such sensor networks are, for example, used as early warning systems for tsunamis[2] and other natural catastrophes. Using an aggregate signature can save bandwidth consumption and increase the performance of the system that evaluates the findings.

Another possible application is *secure logging* [MT08; Har+17a]. Log files are used to monitor events happening to a digital system, like user actions, failed log-in attempts, system errors and other general information. They play an important role in computer security and are, for example, used for intrusion detection. It is therefore necessary that the log is stored securely, which usually involves the application of a digital signature scheme. Since these logs need to be kept for long times and potentially need to store millions of entries, using an aggregate signature scheme can have drastic positive effects on the memory usage and performance of the system.[3]

The authentication of software is another field in which aggregate signatures could be used to great effect [AGH10; Har+16]. In mobile and embedded devices like smart phones, it has become common to sign software to ensure the validity of their code. Very often, vendors also only allow the installation and execution of programs from official app stores. Especially mobile operating systems might only allow signed programs to run. Aggregate signature schemes could be used to improve download bandwidth and decrease the verification overhead.

This thesis discusses two additional properties of aggregate signatures, namely fault-tolerance and deaggregation security. Fault-tolerance protects the scheme against the aggregation of invalid signatures and deaggregation security ensures that attackers cannot remove individual signatures from an aggregate. Both concepts are explained and motivated in detail in the following paragraphs.

1.1.1 Fault-Tolerance

Unfortunately, aggregate signature schemes also have a big disadvantage. Since now only *one* signature is being stored, it is also a single point of failure. It is critical that this signature remains valid, especially when new signatures get aggregated, since it can happen that users and systems aggregate faulty signatures by mistake, as the result of some system error or even with malicious intent.

However, all known aggregate signature schemes have one problem in common: once the aggregate contains even *one single* faulty message-signature pair, the whole aggregate becomes invalid, i.e. they are not *fault-tolerant*. Whether this pair itself was already invalid during aggregation (i.e. the individual signature was not valid for the message) or a "wrong" message is included during verification is insignificant and verification will completely fail in both cases.

All security guarantees for integrity and authenticity are now lost, even if all other messages were correctly and validly signed. Once an invalid message-signature pair is contained in the aggregate, the aggregate signature can no longer be used to infer any information about the integrity and authenticity

[2] See `http://www.ioc-tsunami.org/`. Last accessed February 5, 2020.
[3] See [Har20] for an extensive discussion on secure logging schemes.

of the messages. If such an invalidation of an aggregate signature occurs, the only solution seems to be to sign all data once more. However, this approach has problems of its own. First, this process takes time and in the interim, the security of the data is not guaranteed. Second, depending on the circumstances and applications, signers might no longer be available or even unwilling to sign the same message again. Even worse, this gives signers the opportunity to retract their signature. Suppose, for example, that the scheme is used to sign legally binding documents like contracts. If the aggregate signature becomes invalid, this gives signers the opportunity to *not* sign their contracts again. Since the old signature was lost, there is now no formal way left of proving that the specific party once signed the document.

Many aggregate signature schemes (for example [Lys+04; Nev08; LLY13a; LLY13b]) include the verification step in their aggregation algorithm and only aggregate valid signatures. While this is undesirable, since it introduces additional computational overhead, it intuitively seems to solve this problem.[4] However, this is not the case. This approach only protects against the case of aggregating invalid message-signature pairs. But the same problem occurs if a message is modified *after* its corresponding signature was aggregated. For instance, the message might get changed by an error of the storage medium or transmission errors. Now, the signature of the message becomes invalid "after the fact", so to speak, and the aggregate signature becomes invalid as well.

Let us shortly discuss the implications for some applications of aggregate signature schemes. For sensor networks, if one invalid signature gets included, all findings of the other sensors can no longer be trusted and are therefore lost. For secure logging, if one log entry is not correctly signed, changed or lost (for example through hard disk errors or crashes), the signature for the log becomes invalid. Ma and Tsudik [MT08] actually name this as one of the reasons why they need to store individual signatures for all entries, although they are using an aggregate signature to protect the complete log file. This is undesirable, since it drastically increases the necessary memory space and nullifies one of the advantages offered by aggregate signatures. For software authentication, suppose for example that specific software suites are verified upon the start of the device and only validly signed apps are allowed to run. Now, if one of these signatures becomes faulty, no program can be correctly verified. In the worst case, this could mean that the operating system now blocks the execution of all programs.

1.1.2 Deaggregation Security

Like all types of digital signatures, the security properties of aggregate signature schemes guarantee that forgeries are practically impossible. However, another type of attack is of concern for these schemes: Suppose an attacker gets a hold of an aggregate signature for a set of messages, for example by eavesdropping or actively infiltrating a system. Then the question is whether he can compute a valid signature for a subset of the signed messages without forging a signature by removing or *deaggregating* already aggregated message-signatures pairs from the known aggregate. For example, the attacker might try to selectively remove unwanted entries from a database. One might assume that aggregate signature

[4] It should also be noted that, unfortunately, some schemes critically rely on this verification step while aggregating for their security and can be broken if it is removed [Lys+04; Nev08; BGR14].

schemes are already secure against these types of attacks, since aggregation seems like a one-way operation at first glance. Unfortunately, this is not the case. Moreover, the formal security definition to capture security against forgery attacks also does not imply security against such deaggregation attacks and many schemes only offer limited or even no protection against them.

Such attacks can have drastic effects on applications of aggregate signature schemes. In general, this type of attack can be used to remove data. For example, if an aggregate signature scheme is used to protect a log scheme, an attacker might be able to remove unwanted entries from the log file, like the entries that attest to his break-in. This renders the logging scheme useless, since one of the main reasons to use it is to document attacks. This problem is usually solved by employing some additional mechanisms [MT08; Har+17a; Har20], but it would be advantageous if the used aggregate signature scheme already provides the needed security by itself. For sensor networks, such removal attacks could be used to delete results of the measurements or to hide specific activities. For software authentication, this type of attack could potentially be used to uninstall or remove libraries that are essential to the operation or security of the device. If the scheme is used to sign contracts, such an attack could make it possible for an attacker to void undesired contracts by removing the corresponding message-signature pair from the aggregate.

1.2 Contributions of this Thesis

The contributions of this thesis are as follows:

Fault-Tolerance: To solve the problem that aggregating faulty signatures renders aggregate signatures unusable, we introduce the notion of *fault-tolerance*. A fault-tolerant aggregate signature scheme is able to withstand the inclusion of faulty signatures up to a certain limit. Here, the verification algorithm no longer only outputs a binary value for "valid" or "invalid", but rather a list of messages. Only validly signed messages are part of the output and the security properties guarantee that no invalid message is included in the list.

We give a formal framework of rigorous definitions that capture the fault-tolerance property. If a scheme is fault-tolerant according to our definitions, invalid signatures can be included in an aggregate signature up to a previously *fixed* bound before the verification algorithm can no longer correctly verify the correctly signed messages. We then give the *first* construction of a fault-tolerant aggregate signature scheme. Our construction can be used to turn *any* aggregate signature scheme into a fault-tolerant one. The main building block is a so called "cover-free family" [KS64], which is a combinatorial structure that offers properties that can be elegantly used to achieve fault-tolerance. The construction is *black-box*, i.e. we do not need to assume or exploit any special or additional properties of the underlying aggregate signature scheme or cover-free family to apply our transformation. We formally prove that our construction is secure against forgery. This security proof is *tight*, meaning that applying our transformation incurs no loss in security and the resulting fault-tolerant scheme offers the same level of security against forgery attacks as the underlying aggregate signature scheme.

Our construction also offers the additional feature that it is *selectively verifiable*, meaning it is possible to verify subsets of the signed messages in a manner that is faster then verifying the complete aggregate signature. Essentially, only the parts necessary to verify the subset need to be checked. This offers an additional performance improvement for applications that regularly need to verify subsets.

Our scheme has two restrictions. First, only a fixed number of messages can be aggregated, we however also show how to enhance the scheme to achieve unbounded aggregation. Second, it only offers a slightly restricted form of aggregation. The resulting aggregate signature requires an order among the messages, but *not* an order among the aggregation steps. The signers need to agree upon the "position" of their respective messages in the resulting aggregate, the signatures themselves can then be aggregated in any order, as long as each signer used their assigned "position". The provided form of aggregation is therefore only slightly restricted compared to usual schemes and still flexible enough for many practical applications.

We also investigate the connection between signature size and fault-tolerance and show that, unfortunately, fault-tolerant aggregate signature schemes cannot offer optimal compression, meaning that aggregate signatures cannot be of a constant size.

Furthermore, we describe how to concretely instantiate our scheme using a cover-free family based on polynomials over a finite field [KS64]. Using this cover-free family, we can construct a scheme that has a compact representation featuring short aggregate signatures relative to the number of individual signatures. Signatures only grow logarithmically in the number of messages, which we show to be *optimal*.

We furthermore adapt this framework to a more restricted form of aggregate signatures, called sequential aggregate signatures. Here, for technical reasons, slightly different definitions are needed, but our construction can nevertheless also be used to construct fault-tolerant sequential schemes.

Deaggregation Security: We give several new security definitions of various strength to capture deaggregation security and discuss the relationships between these definitions, to unforgeability and to existing definitions. We show that there exists a true hierarchy of security definitions, meaning that there exist several distinct levels of security against deaggregation attacks ranging from only a basic protection against strongly restricted attackers to definitions that closely model real-life scenarios and that capture very strong deaggregation attackers. However, we also show that several levels of this hierarchy collapse for specific aggregate signature schemes that exhibit three rather natural properties called extendability, claim-removability and order-independence (these properties might also be of independent interest). This means that for schemes that fulfill these properties weak forms of deaggregation security imply stronger definitions.

Furthermore, we discuss the deaggregation (in-) security of a number of existing and important aggregate signature schemes that so far were not examined with deaggregation security in mind. We provide new security proofs and attacks on these schemes and thereby precisely classify their offered deaggregation security and show which level of security in the hierarchy is provided by the respective scheme.

To conclude, we discuss the connection between fault-tolerance and deaggregation security. As argued above, both properties are important and necessary for applications. However, we show that fault-tolerant schemes cannot offer *any* form of deaggregation security and do not fulfill even the weakest definition of our hierarchy. In fact, deaggregation *insecurity* seems to be a virtually "built-in feature" of fault-tolerance and no meaningful form of deaggregation security can be provided by these schemes. This implies that one needs to carefully decide which of the features should be provided directly by the aggregate signature scheme and which property needs to be achieved by implementing other and potentially costly security and safety measures, for example a strong and secure back-up mechanism to protect the application against deletion attacks.

1.3 Structure of this Thesis

Each main chapter contains a summary of the contributions presented in it, as well as a discussion of the related work. This thesis is structured as follows:

- Chapter 2 introduces notation and the basics of cryptography. Note that we only give a brief overview over modern cryptography and only present the topics that are fundamental for understanding this thesis. For further reading, we recommend [Gol01; AB09; Kat10; KL14].
- Chapter 3 gives a detailed introduction to aggregate signature schemes. Fully flexible, sequential and synchronized aggregate signature schemes are introduced and discussed in detail. For each type of aggregation, an overview over the currently known schemes and current state of research is given.
- Fault-Tolerance, the first contribution of this thesis, is discussed in Chapter 4. All results concerning fault-tolerance, like the formal framework and the black-box construction that can be used to turn any aggregate signature scheme into a fault-tolerant one, are presented in this chapter. Note that this chapter is taken almost entirely from publications of the author [Har+16; Har+17a; Har+17b] that were published at the conferences Public Key Cryptography 2016 and Provable Security 2017.
- The second contribution of this thesis, deaggregation security, is discussed in Chapter 5. All results concerning deaggregation security, like the hierarchy of security definitions and the survey of deaggregation security of existing schemes, are presented here. Note that this chapter is in minor parts based on work done together with Roland Gröll [Grö16].
- To conclude, Chapter 6 summarizes the main contributions of this work and discusses future research directions related to this thesis.

Chapter 2

Preliminaries

In this chapter we introduce and define basic cryptographic concepts, primitives, definitions and notations used throughout this thesis. The aim of this chapter is to give a brief overview, for more detailed discussions of the foundations of modern cryptography, we recommend [Gol01; AB09; Kat10; KL14].

2.1 Notation

For $n \in \mathbb{N}$, we write the set $\{1, \ldots, n\}$ as $[n]$, i.e. $[n] \coloneqq \{1, \ldots, n\}$. If M is a set, then $m \leftarrow M$ denotes that m is an element in M, which was uniformly and randomly sampled. For a probabilistic algorithm A, $y \leftarrow A(x)$ means that y is a possible output of A executed on input x. For $k \in \mathbb{N}$, the notation 1^k denotes the unary string of length k, i.e.

$$1^k = \underbrace{1 \ldots 1}_{k \text{ occurrences of } 1}.$$

If M is a multiset and $m \in M$, then we call the number of occurrences of m in M its *multiplicity*. For two multisets M_1, M_2, the union $M_1 \cup M_2$ is defined as the multiset where the multiplicity of each element is the sum of the multiplicities in M_1 and M_2. For example, if $M_1 := \{1, 1, 2\}$ and $M_2 := \{1, 2, 3\}$, then $M_1 \cup M_2 = \{1, 1, 1, 2, 2, 3, 3\}$.

If b is a bit string, then $|b|$ denotes its size, i.e. the number of bits needed to represent b. If v is a vector or a tuple and $i \in \mathbb{N}$, $v[i]$ refers to the i-th entry of v. For matrices M, $\text{rows}(M)$ and $\text{cols}(M)$ denote the number of rows and columns of M, respectively. For $i \in [\text{rows}(M)], j \in [\text{cols}(M)]$, $M[i, j]$ is the entry in the i-th row and j-th column of M.

In this thesis we write groups in multiplicative notation, so if $\mathbb{G}$ is a group, then its group operation is denoted by the common multiplication symbols "$\cdot$" and "$\prod$". If $\mathbb{G}$ is cyclic, then we usually use g to denote a generator of $\mathbb{G}$. Furthermore, σ usually refers to signatures of standard (aggregate) signature schemes, whereas τ mostly refers to signatures of fault-tolerant aggregate signature schemes (signatures, aggregate signatures and fault-tolerant aggregate signatures are discussed in later chapters, see Section 2.2.2, Chapter 3 and Chapter 4). The symbol $\perp$ is used as an error symbol (e.g. if an algorithm needs to abort) or denotes a claim placeholder (see Section 3.1) and λ represents the empty signature (see Definition 3.1.5).

2.2 Basics of Cryptography

As is usual in modern cryptography, we model attackers and cryptographic algorithms as *PPT algorithms*.

Definition 2.2.1 (PPT Algorithm)**.** *An algorithm A is a* probabilistic polynomial time algorithm *(abbreviated as* PPT algorithm*) if the run time of A is polynomial in its input and A is a (possibly) probabilistic algorithm.*

We follow the common asymptotic way of defining security by providing every algorithm with the *security parameter* $\kappa \in \mathbb{N}$ (or more precisely 1^κ) in its input, so that the runtime of each PPT algorithm can be polynomial in this parameter.[1] Throughout this thesis, the symbol κ is used to denote the security parameter. For convenience, we often omit writing the security parameter in the input of an algorithm. If this is the case, it is still assumed that the algorithm implicitly receives 1^κ as part of its input.

The goal of modern cryptographic research is to construct secure schemes and protocols, so that efficient attackers (i.e. PPT algorithms) only have negligible success probability in breaking the scheme. For each cryptographic primitive, we need to formally define what "breaking the scheme" exactly means. This is done by capturing the needed security in *security definitions* and then proving that the scheme in question fulfills this definition by showing that no PPT algorithm has more than negligible success probability in breaking the scheme. We discuss this in more detail for signature schemes in the following sections. The term "negligible" is also precisely defined:

Definition 2.2.2 (Negligible Function)**.** *A function* $f : \mathbb{N} \to [0, 1]$ *is* negligible*, if*

$$\forall c \in \mathbb{N}\ \exists k_0 \in \mathbb{N}\ \forall k > k_0 : |f(k)| \leq \frac{1}{k^c}.$$

It is often colloquially said that a negligible function "asymptotically disappears faster than any polynomial". For example, $2^{-\kappa}$ is negligible, whereas $\kappa^{-100,000}$ is not. In this thesis, we usually denote negligible functions by negl.

2.2.1 Computational Hardness Assumptions & Reductions

Today, the security of cryptographic primitives like encryption schemes or digital signatures is usually based on the *assumed* hardness of well-understood and thoroughly researched complexity theoretical problems like factoring large numbers or calculating the discrete logarithm in a finite algebraic group.

Almost any cryptographic security proof that does not rely on such assumptions would imply $\mathsf{P} \neq \mathsf{NP}$. Since the P versus NP problem is one of the most important, but also most difficult and elusive open questions in computer science, using such assumptions is the only way of proving security considering the current state of research, apart from restrictive and impractical information theoretical constructions.[2]

[1] Algorithms receive 1^k as their input, so that their runtime can actually be polynomial in κ. If the input would be κ instead, their runtimes would only be allowed to be polynomial in $\log(\kappa)$, since κ can represented by $\log(\kappa)$ bits.

[2] See [AB09] for more details on the connection between cryptography, computational complexity and the P versus NP problem.

The security of the cryptographic primitive in question is then *reduced* to the assumed hardness of a computational problem. More concretely, let P be such a problem. It is then assumed that no PPT algorithm can solve P with more than negligible probability. Then the strategy to prove security is usually as follows: Suppose $\mathcal{A}$ is an attacker on the cryptographic primitive. The proof then constructs a new algorithm $\mathcal{B}$ (often called a *simulator*), which simulates the environment that $\mathcal{A}$ needs to run its attack, i.e. $\mathcal{B}$ provides $\mathcal{A}$ with all necessary cryptographic parameters, like public keys, signatures, the used algebraic group and so on. $\mathcal{B}$ does this in such a way that it can "translate" the result of a successful attack of $\mathcal{A}$ into a valid solution for the problem P. By contradiction, it follows that no such attack is possible with more than negligible probability, as long as the assumption that P is difficult to solve holds. Such proofs are usually called *reductions*. See Figure 2.1 below for a schematic representation of how cryptographic reductions work in general.

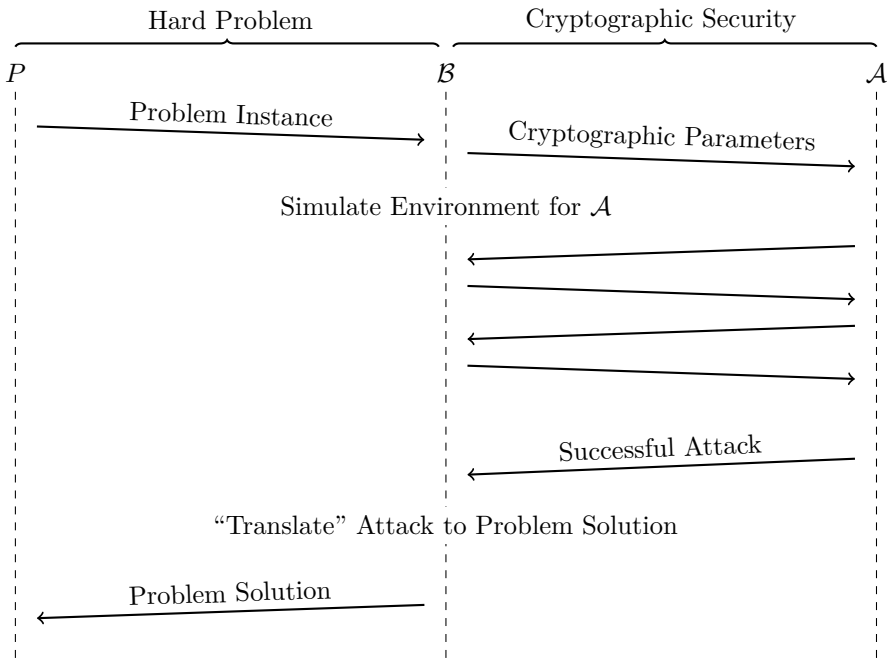

Figure 2.1: Schematic representation of cryptographic reductions.

A very common assumption is the *Computational Diffie-Hellman assumption*, which is based on the groundbreaking work of Diffie and Hellman [DH76].

Definition 2.2.3 (Computational Diffie-Hellman Assumption)**.** *Let $\mathbb{G}$ be a cyclic group of order p depended on the security parameter κ and g a random generator of $\mathbb{G}$. Let $x, y \leftarrow \mathbb{Z}_p$. The* Computational Diffie-Hellman Assumption *states that for all PPT algorithms $\mathcal{A}$ given g, g^x, g^y it is difficult to calculate g^{xy}, i.e. it holds that*

$$\Pr[\mathcal{A}(g, g^x, g^y) = g^z \ : \ g^z = g^{xy}, x, y \leftarrow \mathbb{Z}_p] \leq \mathsf{negl}(\kappa)$$

for a function negl*, which is negligible in the security parameter κ.*

The Computational Diffie-Hellman assumption can also be represented as a *security experiment* between an attacker $\mathcal{A}$ and a challenger $\mathcal{C}_{\mathsf{CDH}}$, where the challenger asks the attacker to solve a random instance of the Computational Diffie-Hellman problem, see Figure 2.2 below.

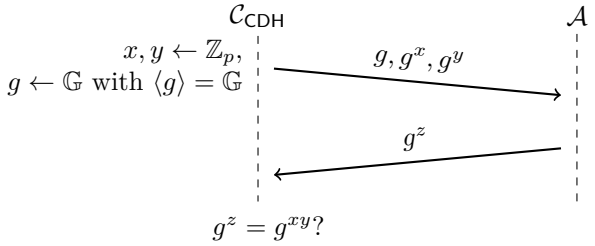

Figure 2.2: The security experiment corresponding to the Computational Diffie-Hellman assumption.

2.2.2 Digital Signatures

Digital signature schemes are a general cryptographic tool used to preserve the *integrity* and *authenticity* of digital documents. Their applications are numerous, ranging from online communication to securing operation system updates. Digital signatures were first proposed by Diffie and Hellman [DH76] and then studied by, for example, [RSA78; Lam79; Rab79; Mer89; Rom90]. The main part of this thesis focuses on a specific type of digital signature, namely so called *aggregate signature schemes*, which are discussed in Chapter 3.

In their seminal work, Goldwasser, Micali, and Rivest [GMR88] gave the first rigorous and formal treatment of digital signatures. They provided a formal definition of digital signatures and their security and also constructed the first scheme satisfying these definitions.

Digital signature schemes work as follows: using their *secret key* sk, a user is able to create signatures σ for messages m (which can be any type of document, like an email, a file, a database etc.), such that it is possible to publicly verify the authenticity of the signature by using a *public key* pk.[3] The goal is to provide users with digital signature schemes for which it is computationally unfeasible to forge signatures without knowing the secret key sk. We now define these concepts formally.

Definition 2.2.4 (Digital Signature Scheme)**.** *A* digital signature scheme *or* digital signature Σ *is a triple* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *of three PPT algorithms as follows:*

- *The key generation algorithm* $\mathsf{Gen}(1^\kappa)$ *receives the security parameter* κ *in unary notation* 1^κ *as its input and outputs a tuple* $(\mathsf{pk}, \mathsf{sk})$ *consisting of the* public key pk *(sometimes also called* verification key*) and the* secret key sk.

[3] Digital signatures are always public key schemes. There is a secret key equivalent, where only one secret key is being used for both computing and verifying signatures. These schemes are usually called *message authentication codes* (MAC). However, we do not discuss these schemes further in this thesis.

- *The signature generation algorithm* $\mathsf{Sign}(\mathsf{sk}, m)$ *receives the secret key* sk *and a message* m *as its input and outputs a signature* σ*.*

- *The verification algorithm* $\mathsf{Vfy}(\mathsf{pk}, m, \sigma)$ *receives the public key* pk*, a message* m *and a signature* σ *as its input and outputs* 0 *or* 1*. The output* 0 *indicates that the signature* σ *was not correct for the given message* m *and therefore could not be verified, whereas* 1 *signifies that* σ *is a correct signature for* m*. If the output is* 1*, we call* σ valid *for the message* m*.*

  Given the same input, the Vfy *algorithm will always output the same value, i.e. its output is* deterministic.[4] *If* $\mathsf{Vfy}(\mathsf{pk}, m, \sigma) = 1$*, we say that the signature* σ *is* valid *for the message* m *under the public key* pk*.*

For digital signature schemes, we require *correctness*, i.e. if a signature for a message is generated by correctly using the Sign algorithm, the Vfy algorithm should output 1.

Definition 2.2.5 (Correctness of Digital Signature Schemes)**.** *A digital signature* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *is* correct*, if for all security parameters* $\kappa \in \mathbb{N}$*, all* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ *and all messages* m *it holds that*

$$\mathsf{Vfy}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1.$$

So far, we have not specified what types of messages can be signed by a digital signature scheme. Throughout this work, if not stated otherwise, we assume that messages are bit strings from a set $\{0,1\}^{p(\kappa)}$, where p is a polynomial in the security parameter κ. The concrete polynomial p is dependent on the respective digital signature scheme. Specific schemes might require messages to be elements of a more precisely defined *message space*, for example an element of a cyclic group. Note that this is only a superficial restriction, since there usually is a direct mapping between bit strings and these types of elements.

The next step is to define what it means for a digital signature scheme to be *secure*. In a security definition, it is of utmost importance to clearly and precisely define the goal of the attack and which means are available to the attacker. The goal of an attack against a digital signature scheme is to forge a valid signature without knowledge of the secret key sk. The attacker is modeled as a PPT algorithm and can therefore compute any operation with probabilistic means in polynomial runtime (in the security parameter κ). The attacker might also receive restricted access to the secret key in form of some type of oracle. The de facto standard security definition for digital signatures is *Existential Unforgeability under Chosen Message Attacks*, abbreviated as $\mathsf{EUF\text{-}CMA}$. Here, the attacker has access to an oracle that provides her with correctly computed signatures on any message of her choosing, modeling the fact that an attacker might observe and influence signatures, for example by eavesdropping on a network or through social engineering. The attack is seen as successful if the attacker can forge a correct signature for *an arbitrary message*, as long as she did not request a signature on this message from the oracle. While there are weaker, as well as stronger, security definitions available for digital signatures,

[4] In most schemes, the Vfy algorithm is completely deterministic. Still, there are some schemes, like [LLY13b], that use randomness in the computation of the Vfy algorithm, which is why we allow it to be a PPT algorithm. However, we require its output to be deterministic, as per the definition above, so that the concept of signatures being *valid* is well defined.

these are usually only interesting from a theoretical viewpoint and EUF-CMA is sufficient and necessary to provide security in many real-life scenarios.

Security definitions then model security as a *security experiment* or *game* between an attacker $\mathcal{A}$ and a challenger $\mathcal{C}$, which represents the attacked scheme and provides $\mathcal{A}$ with all necessary public information and oracle accesses.

Definition 2.2.6 (EUF-CMA Security Experiment)**.** *The* EUF-CMA security experiment *between an attacker $\mathcal{A}$, a challenger $\mathcal{C}$ and a digital signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ consists of three phases as follows:*

Setup Phase. *The challenger $\mathcal{C}$ generates a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ and gives the public key pk to the attacker $\mathcal{A}$.*

Query Phase. *The attacker $\mathcal{A}$ may request signatures for any message from the challenger. If $\mathcal{A}$ sends a message m_i ($i \in \mathbb{N}$) to the challenger, it responds by computing $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$ and sending σ_i to $\mathcal{A}$. $\mathcal{A}$ may repeat this step at will. The amount $q \in \mathbb{N}$ of message requests is only limited in the runtime of $\mathcal{A}$. For PPT algorithms this means that q will be a polynomial $q(\kappa)$ in the security parameter κ.*

Forgery Phase. *At the end of the experiment, $\mathcal{A}$ sends a tuple (m^*, σ^*) consisting of a message m^* and a signature σ^* to the challenger $\mathcal{C}$. $\mathcal{A}$ is* successful*, if*

$$\mathsf{Vfy}(\mathsf{pk}, m^*, \sigma^*) = 1 \quad \textit{and} \quad m^* \notin \{m_1, \ldots, m_q\}.$$

According to the definition, an attacker $\mathcal{A}$ is successful, if she is able to forge a correct signature σ^* for any message m^*. The second restriction ensures that trivial forgeries are not interpreted as successful, i.e. the attacker does not win if she outputs a signature for a message she requested a signature on from the challenger. See Figure 2.3 for a visualization of the EUF-CMA security experiment.

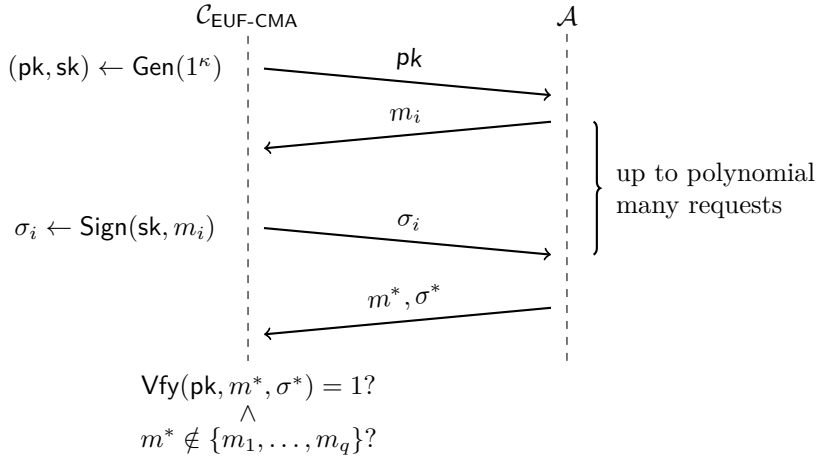

Figure 2.3: The EUF-CMA security experiment.

Definition 2.2.7 (EUF-CMA Security for Digital Signature Schemes)**.** *A digital signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *is* EUF-CMA secure*, if all PPT algorithms* $\mathcal{A}$ *only have negligible success probability in the* EUF-CMA *security experiment, meaning it holds that*

$$\Pr\left[\mathcal{A}^{\mathcal{C}_{\mathsf{EUF\text{-}CMA}}}(\mathsf{pk}) = (m^*, \sigma^*) \ : \ \begin{array}{c} \mathsf{Vfy}(\mathsf{pk}, m^*, \sigma^*) = 1 \\ \wedge\, m^* \notin \{m_1, \ldots, m_q\} \end{array}\right] \leq \mathsf{negl}(\kappa)$$

for a function negl*, which is negligible in the security parameter* κ*.*

Another common and more precise way of defining security is as follows:

Definition 2.2.8 (EUF-CMA Security – Concrete Formulation)**.** *A digital signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *is* (t, q, ϵ)-EUF-CMA secure*, if there exists no attacker* $\mathcal{A}$ *running in time at most* t*, making at most* q *queries to the signature oracle and that wins the* EUF-CMA *security experiment with success probability at least* ϵ*.*

This type of definitions is sometimes called "concrete security", since the theorems and definitions explicitly state the runtime and success probability of the attacker, instead of using more abstract terms like "negligible" and "polynomial".

The advantage of this type of definition is that it gives a measure of the quality of cryptographic security reductions. It allows to clearly show the dependencies between the runtime and success probability of the constructed simulator $\mathcal{B}$ of the reduction and the assumed attacker $\mathcal{A}$. Ideally, the runtime and success probability of $\mathcal{A}$ and $\mathcal{B}$ are almost identical. If this is the case, the reduction is called *tight*. However, the runtime of $\mathcal{B}$ might be significantly higher than the runtime of $\mathcal{A}$, while still being polynomial. The same applies for the success probability of $\mathcal{B}$, which might be significantly lower than the success probability of $\mathcal{A}$. This has a direct influence on the choice of the size of the security parameter. The farther apart the runtimes and success probabilities are, the larger the security parameter needs to be for the scheme to provide security in practical applications, which directly influences the size of the keys and runtimes of the algorithms of the scheme and therefore its efficiency. See, for example, [Bad+16] for a discussion on cryptographic tightness.

Note however that the difference between these two types of definitions is only the presentation and concreteness of their statements. Conceptually there is no difference: for a scheme to be considered secure, it is always required that the success probability of all PPT attackers must be negligible.

For ease of presentation, and since most proofs in this thesis do not profit from this type of notation, we have chosen to present almost all theorems and definitions in the more abstract way as in Definition 2.2.7. The exception is Chapter 4, were we present our constructions to achieve fault-tolerance. Here, we use the concrete type to be consistent with the original publication of the construction [Har+16; Har+17a].

2.2.3 Hash Functions and the Random Oracle Model

Hash functions are an important cryptographic tool for computing short and fixed-length "fingerprints" for inputs of any length and are often used in conjunction with digital signature schemes.

Definition 2.2.9 (Hash Function)**.** *A* hash function H *is a tuple of two PPT algorithms* $H = (\mathsf{Gen}, \mathsf{Eval})$ *as follows:*

- $\mathsf{Gen}(1^\kappa)$ *receives as input the security parameter* κ *and outputs a key* t *which describes a function*

$$H_t : \{0,1\}^* \to \mathcal{M}_t,$$

  where M_t *is a finite set, which is also defined by* t*.*

- $\mathsf{Eval}(1^\kappa, t, m)$ *receives as input the security parameter* κ*, a key* t *and a message* m *and computes* $H_t(m) \in \mathcal{M}_t$*.*

For convenience, we usually omit the key t and write $H(m)$ as shorthand for $H_t(m)$, where t was generated by $\mathsf{Gen}(1^\kappa)$. Furthermore, it is common to write $H(a, b, \ldots, z)$ to denote $H(a\|b\|\ldots\|z)$, i.e. to specify multiple inputs to the hash function, although formally only one input is defined.

We usually require hash functions to be collision resistant, meaning that it should be difficult to compute values $x \neq x'$ which evaluate to the same hash value, i.e. $H(x) = H(x')$.

Definition 2.2.10 (Collision Resistance)**.** *A hash function* $H = (\mathsf{Gen}, \mathsf{Eval})$ *is* collision resistant*, if for all* $t \leftarrow \mathsf{Gen}(1^\kappa)$ *and for all PPT algorithms* $\mathcal{A}$ *it holds, that*

$$\Pr[\mathcal{A}(1^\kappa, t) = (x, x') \,:\, H_t(x) = H_t(x') \wedge x \neq x'] \leq \mathsf{negl}(\kappa)$$

for a function negl*, which is negligible in the security parameter* κ*.*

See Figure 2.4 for a visualization of the security experiment implicitly given in the definition of collision resistance.

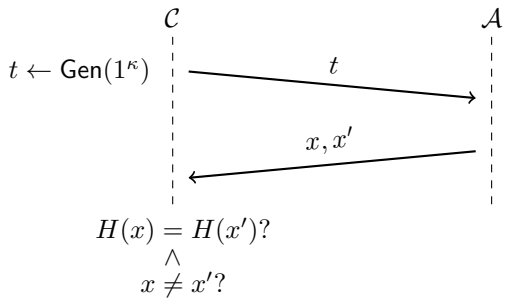

Figure 2.4: The collision resistance security experiment.

A common technique is to combine digital signatures and hash functions by signing the hash value of the message instead of the message itself, i.e. computing $\mathsf{Sign}(\mathsf{sk}, H(m))$ instead of $\mathsf{Sign}(\mathsf{sk}, m)$. This makes it possible to sign messages which are longer then the messages in the message space $\{0,1\}^{p(\kappa)}$ of the signature scheme in a secure manner. It can also have a positive effect on the performance of the application using the signature scheme. If the signature scheme is EUF-CMA secure and the hash function is collision resistant, then this combination is also EUF-CMA secure. This technique is often called the "Hash-then-Sign paradigm".

The *random oracle model* [BR93] is an idealization of hash functions. Here, the hash function has no source code and users (attackers, algorithms etc.) cannot evaluate the hash function themselves, i.e. there is no program or algorithm available which can be used to compute $H(x)$. Instead, all participants have oracle access to a *random oracle*, which works as follows: If a hash value $H(x)$ for input x is to be computed, the user sends x to the random oracle. The oracle then checks whether it already output a value $H(x)$ for x at a previous time and if so, it outputs this stored value. If not, a truly random string from the output space of the hash function is chosen, saved as $H(x)$ and returned to the caller of the oracle.

Cryptographic schemes which are proven secure by using random oracles are said to be secure in the *random oracle model*. In contrast, proofs in the *standard model* of cryptography do not use or rely on random oracles.

The random oracle model enables simple, elegant and very efficient cryptographic constructions, for example [BR94; BR96; Bon+03; BLS04]. Nonetheless, the random oracle model is controversial, since there exists no such oracle and schemes using a random oracle need to be instantiated with "normal" hash functions, if they are to be used in real-world applications. It is therefore unclear how results proven in the random oracle model transfer to the real world and what security guarantees can be inferred from random oracle model proofs. Furthermore, there are constructions which can be proven secure in the random oracle model, but are also proven to be insecure if *any* real hash function is used to instantiate them [CGH04]. However, these constructions are purpose-built to break down like this, which is why some researchers argue that they are only pathological and contrived. Moreover, there also exists cryptographic primitives that can *only* be constructed in the random oracle model, see for example [Nie02]. Koblitz and Menezes [KM15] offer a detailed survey and discussion of the random oracle model and its applications and problems.

2.2.4 Pairings

A *pairing* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map from two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to a so-called target group $\mathbb{G}_T$. Because of their bilinearity, they allow a restricted form of multiplication in the exponent, since $e(g^x, g^y) = e(g, g)^{xy}$, which is not possible in groups without pairings. This seemingly simple operation is a powerful tool with many cryptographic applications. A small selection of examples can be found in [MOV93; Bon+03; BLS04; Coh+05; Wat05; GPS08].

Definition 2.2.11 (Pairing)**.** *Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be cyclic groups of order p.*[5] *A* pairing *is a map*

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

which satisfies the following properties:

Bilinearity. *For all $g_1, g_1' \in \mathbb{G}_1$ and all $g_2, g_2' \in \mathbb{G}_2$ it holds that*

$$e(g_1 \cdot g_1', g_2) = e(g_1, g_2) \cdot e(g_1', g_2),$$
$$e(g_1, g_2 \cdot g_2') = e(g_1, g_2) \cdot e(g_1, g_2').$$

Non-Degenerate. *For all generators g_1 of $\mathbb{G}_1$ and g_2 of $\mathbb{G}_2$, it holds that $e(g_1, g_2)$ is a generator in $\mathbb{G}_T$. Note that if p is prime, this is equivalent to requiring that $e(g_1, g_2) \neq 1$.*

Efficiency. *The map e can be computed efficiently, i.e. in polynomial time in the security parameter κ.*

If $\mathbb{G}_1 = \mathbb{G}_2$, then the pairing is called *symmetric*, otherwise it is called *asymmetric*. For asymmetric pairings, it is also important to know whether there exists a non-trivial and efficiently computable homomorphism between $\mathbb{G}_2$ and $\mathbb{G}_1$. See [GPS08] for an overview of the different types of pairings.

Remark. For simplicity's sake, we present all schemes using pairings in this thesis in the symmetric setting, although many of the schemes can also be instantiated in the more general asymmetric setting. We refer the interested reader to the original publications cited for each scheme for details.

In security proofs for schemes using pairings, the following lemma can be useful:

Lemma 2.2.12. *Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a pairing and $g \in \mathbb{G}$ be a generator of $\mathbb{G}$. Then the functions*

$$f_1 : \mathbb{G} \to \mathbb{G}_T,\ x \mapsto e(g, x)$$
$$f_2 : \mathbb{G} \to \mathbb{G}_T,\ x \mapsto e(x, g)$$

are bijective.

Proof. We only prove the lemma for f_1, the proof for f_2 is analogous. Since $|\mathbb{G}| = |\mathbb{G}_T|$, it suffices to show that f_1 is injective. Let $x, y \in \mathbb{G}$. Since g is a generator, there exists $a, b \in \mathbb{Z}_p$ such that $x = g^a$ and $y = g^b$. Suppose that we have $f_1(x) = f_1(y)$. Then it follows that

$$e(g, g)^a = e(g, g^a) = f_1(x) = f_1(y) = e(g, g^b) = e(g, g)^b,$$

which implies that $a = b$, since $e(g, g)$ is a generator of $\mathbb{G}_T$, according to the definition of pairings. It follows that x and y must be equal, which implies the injectivity and therefore the bijectivity of f_1. □

[5] There are also constructions using pairings over groups of composite order, see for example [BGN05; Gui13]. However, in this thesis, we only use pairings over groups with prime order.

2.3 Cover-Free Families

Cover-free families are combinatorial structures that were first introduced by Kautz and Singleton [KS64] and have various applications in cryptography, for example in traitor tracing, signatures and encryption [SW99; LVY01; TS06; Cra+07; HJK11]. We use cover-free families for our construction of a fault-tolerant aggregate signature scheme in Section 4.5. Using a d-cover-free family allows us to detect and tolerate the aggregation of up to d invalid signatures.

Definition 2.3.1 (Cover-Free Family)**.** *For $d \in \mathbb{N}$, a* d-cover-free family $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$ *(abbreviated as* d-CFF*) consists of a set $\mathcal{S}$ (called* universe*) of $m \in \mathbb{N}$ elements and a set* Blocks *of $n \in \mathbb{N}$ subsets of $\mathcal{S}$, where $d < m < n$, such that: For any d subsets $B_1, \ldots, B_d \in \mathsf{Blocks}$ and all distinct $B \in \mathsf{Blocks} \setminus \{B_1, \ldots, B_d\}$, it holds that*

$$\left| B \setminus \bigcup_{k=1}^{d} B_k \right| \geq 1.$$

So, for a d-cover-free family it is not possible to completely "cover" all elements of a single subset with at most d different subsets of the family. This feature is instrumental in our construction of fault-tolerant aggregate signatures.

Incidence matrices simplify the handling and representation of cover-free families:

Definition 2.3.2 (Incidence Matrix of a Cover-Free Family)**.** *For a d-CFF $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$, where the elements of $\mathcal{S}$ and* Blocks *have a well-defined order, such that we can write $\mathcal{S} = \{s_1, \ldots, s_m\}$, $\mathsf{Blocks} = \{B_1, \ldots, B_n\}$, we define its* incidence matrix $\mathcal{M}$ *as follows:*

$$\mathcal{M}[i,j] = \begin{cases} 1, & \text{if } s_i \in B_j, \\ 0, & \text{otherwise.} \end{cases}$$

The i-th row *of $\mathcal{M}$ is denoted by $\mathcal{M}_i \in \{0,1\}^n$, for $i \in [m]$.*

Each $s_i \in \mathcal{S}$ corresponds to row i and each $B_j \in \mathsf{Blocks}$ corresponds to column j, i.e. $\mathcal{M}$ has m rows and n columns.

Example 2.3.3. Let $\mathcal{S} = \{a, b, c, d\}$ and $\mathsf{Blocks} = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{c, d\}\}$. Then $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$ is a 1-cover-free family, since no set completely covers any other set. However, it is not a 2-cover-free family, because the set $\{a, b\} \cup \{c, d\}$ covers all other sets in Blocks. Its incidence matrix $\mathcal{M}$ is

$$\mathcal{M} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Chapter 3

Aggregate Signatures

In this chapter, we introduce the type of digital signature which this thesis focuses on, namely *aggregate signatures*. Aggregate signature schemes were first proposed and constructed by Boneh, Gentry, Lynn and Shacham [Bon+03] and offer the additional functionality of compressing or *aggregating* several signatures into one short signature.

Suppose there are n different users with n key pairs $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_n, \mathsf{sk}_n)$ of the same aggregate signature scheme and each user has computed a signature σ_i for a message m_i. Then it is possible to aggregate the signatures $\sigma_1, \ldots, \sigma_n$ into one single and short aggregate signature σ_{Agg}, which has the same size as an individual signature for a single message, i.e. $|\sigma_{\mathsf{Agg}}| = |\sigma_i|$.

This aggregate signature σ_{Agg}, together with the respective messages and public keys, is sufficient to convince a verifier that the n users actually signed the n messages. Since the size of σ_{Agg} is smaller then the sum of all sizes of $\sigma_1, \ldots, \sigma_n$ and verifying it is often faster than verifying n individual signatures, using an aggregate signature scheme can have drastic improvements on performance, bandwidth and memory use in many applications.

Since the seminal work of Boneh, Gentry, Lynn and Shacham [Bon+03], this type of digital signature has gathered much interest and several different types of aggregation have been proposed, most importantly *fully flexible*, *sequential* and *synchronized* aggregate signature schemes, which we discuss in detail in the following sections.

3.1 Claims and Claim Sequences

As a notational convenience and to be able to present our concepts in a clear manner, we use *claims* and *claim sequences* as introduced in our publication [Har+16]. A claim (pk, m) conveys the meaning that the owner of pk has signed the message m. A signature σ for m under pk can therefore be interpreted as a proof of the claim. Claims allow for a more compact representation of the algorithms of an aggregate signature scheme, since here we have to deal with multiple signers, public keys and messages.

Definition 3.1.1 (Claim)**.** *A* claim *is a tuple* (pk, m) *consisting of a public key* pk *of a signature scheme and a message* m*.*

Definition 3.1.2 (Claim Sequences)**.** *A* claim sequence *is a tuple $C = (c_1, \ldots, c_n)$ of claims c_i and the placeholder symbol $\perp$, used to denote an empty claim. We use the following notation for claim sequences $C = (c_1, \ldots, c_n)$ and $C' = (c'_1, \ldots, c'_m)$:*

- $\mathsf{elem}(C)$ *denotes the multiset of elements of C excluding $\perp$.*
- $C \setminus C'$ *denotes the claim sequence where all claims of C' are removed from C, i.e. $C \setminus C'$ is the sequence containing the claims of $\mathsf{elem}(C) \setminus \mathsf{elem}(C')$.*
- $C \| C'$ *denotes the claim sequence $(c_1, \ldots, c_n, c'_1, \ldots, c'_m)$.*
- $|C|$ *denotes the number of claims in the sequence, i.e. here $|C| = n$ and $|C'| = m$.*

The placeholder symbol $\perp$ is allowed in claim sequences for technical reasons, which are made clear in Chapter 4. However, for the most part, verification algorithms can simply ignore the placeholders in a claim sequence C and verify the signature on the elements in $\mathsf{elem}(C)$. An important concept for our fault-tolerant construction in Section 4.5 is the *mergeability* of claim sequences.

Definition 3.1.3 (Mergeability and Complements of Claim Sequences)**.** *Two claim sequences C_1 and C_2 are* mergeable*, if for all $i \in [\min(|C_1|, |C_2|)]$ it holds that $C_1[i] = \perp$ or $C_2[i] = \perp$ or $C_1[i] = C_2[i]$. C_1 and C_2 are called* exclusively mergeable*, if for all such i it holds that $C_1[i] = \perp$ or $C_2[i] = \perp$. In particular, two exclusively mergeable sequences are mergeable.*

We denote the merging of two claim sequences C_1 and C_2 with the symbol $\sqcup$. Let C_1 and C_2 be two mergeable claim sequences of length k and l, respectively. Without loss of generality, let $k \geq l$. Then the merged claim sequence *$C_1 \sqcup C_2$ is $(c_1, \ldots, c_k)$, where*

$$c_i \coloneqq \begin{cases} C_1[i], & \text{if } C_2[i] = \perp,\ C_2[i] = C_1[i] \text{ or } i > l, \\ C_2[i], & \text{otherwise.} \end{cases}$$

Example 3.1.4. Let c_1, c_2, and c_3 be three distinct claims and let $C_1 = (\perp, c_2, c_3)$, $C_2 = (c_1, \perp, \perp)$, $C_3 = (c_1, c_2, \perp)$ and $C_4 = (c_1, c_3, c_2)$ be four claim sequences. Then C_1, C_2 are exclusively mergeable, C_1, C_3 are mergeable, but *not* exclusively mergeable, and C_1, C_4 are not mergeable.

Also for technical reasons made clear in Chapter 4, we define the *empty signature*:

Definition 3.1.5 (Empty Signature)**.** *The* empty signature *λ is a signature valid for exactly the claim sequences containing only $\perp$ and the empty claim sequence.*[1]

Definition 3.1.6 (Subsequences of Claim Sequences)**.** *Let $C = (c_1, \ldots, c_n)$ be a tuple and $b \in \{0,1\}^n$ be a bit sequence specifying a selection of indices. Then $C[b]$ is the* subsequence *of C containing exactly the elements c_j where $b[j] = 1$, replacing all other claims by $\perp$. In particular, if $\mathcal{M}$ is a bit matrix (for example an incidence matrix of a cover-free family), then $C[\mathcal{M}_i]$ is the subsequence containing all c_j where $\mathcal{M}[i, j] = 1$ and $\perp$ at all other positions.*

Example 3.1.7. Let $C = (c_1, c_2, c_3)$ be a claim sequence. Then $C[(1,0,0)]$ is equal to $(c_1, \perp, \perp)$ and $C[(0,1,1)]$ is equal to $(\perp, c_2, c_3)$.

[1] To exclude trivial attacks, throughout this thesis all unforgeability definitions do not interpret $(\perp, \ldots, \perp), \lambda$ as a successful forgery and we don't explicitly state this in the definitions.

3.2 Fully Flexible Aggregate Signatures

Fully flexible aggregate signature schemes were introduced in the seminal work of Boneh, Gentry, Lynn and Shacham [Bon+03]. These schemes offer the most flexible type of aggregation: aggregation is a public operation, so no secret key or other secret information is needed to aggregate signatures, it is possible to aggregate signatures in *any* order and already aggregated signatures can always be aggregated further with other aggregate or individual signatures. Unfortunately, this big degree of flexibility in aggregation has also made it difficult to find suitable and efficient schemes, especially in the standard model. Section 3.2.2 gives an overview of the state of research.

Definition 3.2.1 (Aggregate Signature Scheme)**.** *A* fully flexible aggregate signature scheme*, or simply* aggregate signature scheme[2]*, is a tuple* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *of four PPT algorithms as follows:*

- *The key generation algorithm* $\mathsf{Gen}(1^\kappa)$ *receives the security parameter* κ *as its input and outputs a tuple* $(\mathsf{pk}, \mathsf{sk})$ *consisting of the* public key pk *and the* secret key sk*.*

- *The signature generation algorithm* $\mathsf{Sign}(\mathsf{sk}, m)$ *receives the secret key* sk *and a message* m *as its input and outputs a signature* σ*.*

- *The aggregation algorithm* $\mathsf{Agg}(C_1, C_2, \sigma_1, \sigma_2)$ *takes as input two claim sequences* C_1 *and* C_2*, as well as corresponding signatures* σ_1 *and* σ_2 *(note that if* $|C_i| > 1$*, then* σ_i *will be an aggregated signature) and creates an aggregate signature* σ_{Agg}*, certifying the validity of all claims in* C_1 *and* C_2*.*

- *The verification algorithm* $\mathsf{Vfy}(C, \sigma)$ *receives a claim sequence* C *and a signature* σ *as input. It outputs* 1 *if the signature* σ *is valid for* C *and* 0 *otherwise.*

  Like for digital signature schemes (see Definition 2.2.4) we again allow the Vfy *algorithm to use randomness, but expect its output given a fixed input to be deterministic.*

Note that according to our syntax, the Agg algorithm takes two claim *sequences* as input. So, if a single claim c is to be aggregated, it first needs to be converted to a claim sequence $C = (\perp, \ldots, \perp, c)$, with some or no claim placeholders $\perp$ in front of c, depending on the scheme. Since this conversion is usually rather straightforward, we do not pay much attention to it, unless necessary, like for our fault-tolerant construction in Chapter 4.

Furthermore, the order of the claims in a claim sequence C is of no importance for most schemes, i.e. a signature that is valid for a claim sequence C is also valid for all reorderings of the sequence. In this thesis, we usually assume that this is the case for fully flexible schemes, but point out if definitions or theorems need to be adjusted for schemes where the order of claims is actually significant. We require correctness as follows.

[2] We use the terms *fully flexible aggregate signature scheme* and *aggregate signature scheme* interchangeably in this thesis.

Definition 3.2.2 (Correctness of Aggregate Signature Schemes)**.** *An aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *is* correct*, if for all security parameters* $\kappa \in \mathbb{N}$*, all claim sequences* C *consisting of claims* $c_i = (\mathsf{pk}_i, m_i)$ *where* pk_i *was output by* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\kappa)$*, and all signatures* σ *that were created by correct applications of the* Sign *and* Agg *algorithms to create an aggregate signature on all claims of* C*, it holds that* $\mathsf{Vfy}(C, \sigma) = 1$*.*

Remark. The definitions above make no mention of the size of aggregate signatures and therefore technically do not exclude trivial constructions like simply adding individual signatures to a list. However, it usually is expected that the size of an aggregate signature is equal to the size of a single individual signature [Bon+03; HSW13]. The same is true for the performance of the Vfy algorithm. It is expected that verifying an aggregate signature is at least as fast as verifying all individual signatures for all messages, although the definitions also do not formally require this.

The security definition for aggregate signatures given by Boneh et al. [Bon+03] is very similar to EUF-CMA. The main change is that the attacker may also output an aggregate signature as his forgery for a claim sequence of his choosing, which however must contain at least one new claim using the public key issued by the challenger. All other keys of the claim sequence may be freely chosen by the attacker. See Figure 3.1 for a visualization of the security experiment.

Definition 3.2.3 (AS-EUF-CMA Security Experiment)**.** *The* AS-EUF-CMA security experiment[3] *between an attacker* $\mathcal{A}$*, a challenger* $\mathcal{C}$ *and an aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *consists of three phases as follows:*

Setup Phase. *The challenger* $\mathcal{C}$ *generates a key pair* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ *and gives the public key* pk *to the attacker* $\mathcal{A}$*.*

Query Phase. *The attacker* $\mathcal{A}$ *may request signatures for messages of his choosing from the challenger. If* $\mathcal{A}$ *sends a message* m_i *(*$i \in \mathbb{N}$*) to the challenger, he responds by computing* $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$ *and sending* σ_i *to* $\mathcal{A}$*. This step may be repeated at will by* $\mathcal{A}$*.*

Forgery Phase. *At the end of the experiment,* $\mathcal{A}$ *sends a tuple* (C^*, σ^*) *consisting of a claim sequence* C^* *and a signature* σ^* *to the challenger. Let* $q \in \mathbb{N}$ *be the number of messages* $\mathcal{A}$ *queried a signature for in the Query Phase. Then* $\mathcal{A}$ *is* successful*, if*

$$\mathsf{Vfy}(C^*, \sigma^*) = 1 \quad \text{and} \quad \exists(\mathsf{pk}, m^*) \in C^* \quad \text{with} \quad m^* \notin \{m_1, \ldots, m_q\}.$$

Some schemes impose specific restrictions on their use to be secure, for example that all messages in the claim sequence of an aggregate signature must be distinct or that each user may only aggregate one signature, i.e. all public keys in the claim sequence must be distinct.

These restrictions are usually only minor, only have small effects on their applications and can very often be circumvented by additional steps, for example by signing $\mathsf{pk}_i \| m_i$ instead of m_i if all message must be distinct.

For convenience and simplicity's sake, we do not define different security definitions or versions of aggregate signature schemes to reflect these restrictions,

[3] The AS in AS-EUF-CMA stands for "Aggregate Signature".

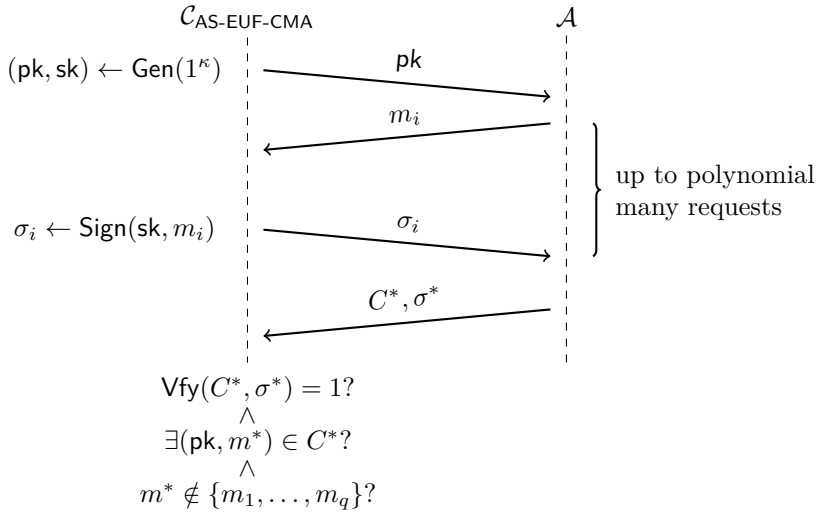

Figure 3.1: The AS-EUF-CMA security experiment.

but rather interpret them as a restriction of the scheme in question and not of the security definition. If necessary, we state these restrictions in the description of the schemes and security theorems. Bellare, Namprempre, and Neven [BNN07] discuss such restrictions, their implications and how to overcome them in detail.

Definition 3.2.4 (AS-EUF-CMA Security for Aggregate Signature Schemes)**.** *An aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *is* AS-EUF-CMA secure*, if all PPT algorithms* $\mathcal{A}$ *only have negligible success probability in the* AS-EUF-CMA *security experiment, meaning it holds that*

$$\Pr\left[\mathcal{A}^{\mathcal{C}_{\mathsf{AS\text{-}EUF\text{-}CMA}}}(\mathsf{pk}) = (C^*, \sigma^*) : \begin{array}{r} \mathsf{Vfy}(C^*, \sigma^*) = 1 \\ \wedge\, \exists(\mathsf{pk}, m^*) \in C^* \\ \wedge\, m^* \notin \{m_1, \ldots, m_q\} \end{array}\right] \leq \mathsf{negl}(\kappa)$$

for a function negl*, which is negligible in the security parameter* κ*.*

3.2.1 The BGLS Aggregate Signature Scheme

We now give a brief overview of the aggregate signature scheme of Boneh, Gentry, Lynn and Shacham [Bon+03]. The scheme is elegant and relatively simple, which is why we use it in the following chapters to illustrate different concepts and examples. We present it using a symmetric pairing, although the scheme also works with asymmetric pairings and is presented in the asymmetric setting in the original publication.

Definition 3.2.5 (BGLS Aggregate Signature Scheme)**.** *Let* $\mathbb{G}, \mathbb{G}_T$ *be cyclic groups of prime order* $p \in \mathbb{N}$*,* g *a random generator of* $\mathbb{G}$*,* $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ *a pairing and* $H : \{0,1\}^* \to \mathbb{G}$ *a hash function. The* BGLS *aggregate signature scheme* $\Sigma_{\mathsf{BGLS}} = (\mathsf{Gen}_{\mathsf{BGLS}}, \mathsf{Sign}_{\mathsf{BGLS}}, \mathsf{Agg}_{\mathsf{BGLS}}, \mathsf{Vfy}_{\mathsf{BGLS}})$ *consists of four PPT algorithms as follows:*

$\mathsf{Gen}_{\mathsf{BGLS}}(1^\kappa)$. *Pick a random* $x \leftarrow \mathbb{Z}_p$ *and compute* g^x. *Return* $(\mathsf{pk}, \mathsf{sk}) := (g^x, x)$.

$\mathsf{Sign}_{\mathsf{BGLS}}(\mathsf{sk}, m)$. *Return* $\sigma := H(m)^{\mathsf{sk}} = H(m)^x \in \mathbb{G}$.

$\mathsf{Agg}_{\mathsf{BGLS}}(C_1, C_2, \sigma_1, \sigma_2)$. *Return* $\sigma_{\mathsf{Agg}} = \sigma_1 \cdot \sigma_2$.

$\mathsf{Vfy}_{\mathsf{BGLS}}(C, \sigma)$. *Parse* $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$. *Ensure that all messages* m_i *are distinct. If not, output* 0. *If it holds that*

$$e(\sigma, g) = \prod_{i=1}^{n} e(H(m_i), \mathsf{pk}_i)$$

return 1, *otherwise return* 0.

Remark. The BGLS aggregate signature scheme is only AS-EUF-CMA secure if all messages for a given aggregate signature are distinct. This additional requirement is enforced by the $\mathsf{Vfy}_{\mathsf{BGLS}}$ algorithm. As already discussed in the remark below Definition 3.2.3, this is only a slight restriction, since this problem can be solved by signing $\mathsf{pk}_i \| m_i$ instead of m_i. Strictly speaking, the $\mathsf{Vfy}_{\mathsf{BGLS}}$ algorithms would still need to check that all $\mathsf{pk}_i \| m_i$ are distinct for the security proof of [Bon+03] to go through, but Bellare, Namprempre, and Neven [BNN07] formally prove that this check can safely be removed.

All signatures are group elements in $\mathbb{G}$ and therefore have the same size, namely the size needed to encode one element of the group $\mathbb{G}$ as a bit string. Aggregation is done by simply multiplying group elements. Note that even though the Agg algorithm is only stated for two signatures, it is possible to aggregate more than two signatures at the same time by simply multiplying all signatures that are to be aggregated. Observe furthermore that the Agg algorithm neither verifies the signatures nor uses the claim sequences in any way. So, in practice, the parties that aggregate signatures, like sensors in a sensor network, would not even need to be given the respective claim sequences, which can further improve bandwidth usage and performance.

Theorem 3.2.6. *If the Computational Diffie-Hellman assumption holds and* H *is modeled as a random oracle, then the* BGLS *aggregate signature scheme is* AS-EUF-CMA *secure in the random oracle model.*

Proof. A proof of this theorem can be found in [Bon+03]. □

3.2.2 Overview of Known Fully Flexible Schemes

The first aggregate signature scheme was proposed by Boneh, Gentry, Lynn and Shacham [Bon+03] (see the previous Section 3.2.1), which can be proven secure under the Computational Diffie-Hellman assumption in the random oracle model. Bellare, Namprempre, and Neven [BNN07] improve on [Bon+03] by formally showing how the restriction that only signatures of distinct messages can be aggregated can be securely removed.

Unfortunately, constructing *efficient* aggregate signature schemes that are fully flexible and secure in the standard model has turned out to be a difficult endeavor and to the best of our knowledge no such scheme is known at the time of publication of this thesis.

Rückert and Schröder [RS13] give a secure construction using multilinear maps in the certified-key model without random oracles, where the attacker needs to prove knowledge of the secret keys corresponding to the public keys used in his oracle queries. They also need to use an interactive assumption to prove their scheme secure. Hohenberger, Sahai, and Waters [HSW13] give a construction of an aggregate signature scheme in the standard model using the graded multilinear maps of Garg, Gentry, and Halevi [GGH12]. Hohenberger, Koppula, and Waters [HKW15] propose so-called *universal signature aggregators*, which can aggregate signatures of *different* signature schemes. They give constructions in both models based on very strong cryptographic primitives like indistinguishable obfuscation.

Several restrictions of aggregation have been proposed. The most prominent are *sequential* and *synchronized* aggregate signature schemes, which are discussed in detail in Section 3.3 and Section 3.4.

3.3 Sequential Aggregate Signatures

Sequential aggregate signatures were first proposed by [Lys+04] and are a variation of aggregate signatures that only offer restricted aggregation capabilities. As the name implies, it is only possible to aggregate in a sequential order.

The first party creates the first signature σ_1 for its message using its own secret and public keys. After σ_1 was created, it then passes on this signature to the next party, who can then aggregate its message to this signature and so on, i.e. signing and aggregating describes a chain of different parties adding their message to the signature one after another.

In particular, it is *not* possible to aggregate different signatures themselves, but only to "add" new messages in a sequential order to an existing signature. Observe that this also implies that aggregation is no longer a fully public operation, since the signing and aggregation operations are closely entwined and are represented by one single algorithm $\mathsf{AggSign}$. Despite these restrictions, the functionality provided by sequential schemes is sufficient for many real-world problems [Lys+04]. So far, they are also more efficient and easier to construct than more flexible schemes.

The definitions and security experiments for sequential aggregate signatures are very similar to those for aggregate signature schemes, with the exception that there is only one single algorithm $\mathsf{AggSign}$ used for signing and aggregating and slight changes to reflect the sequential nature of aggregation.

Definition 3.3.1 (Sequential Aggregate Signature Scheme)**.** *A* sequential aggregate signature scheme *is a tuple* $\Sigma = (\mathsf{Gen}, \mathsf{AggSign}, \mathsf{Vfy})$ *of three PPT algorithms as follows:*

- *The key generation algorithm* $\mathsf{Gen}(1^\kappa)$ *receives the security parameter* κ *as its input and outputs a tuple* $(\mathsf{pk}, \mathsf{sk})$ *consisting of the* public key pk *and the* secret key sk*.*

- *The signature generation and aggregation algorithm* $\mathsf{AggSign}(\mathsf{sk}, C, \sigma, m)$ *receives the secret key* sk*, a claim sequence* C*, a corresponding signature* σ *and a message* m *as its input and outputs a signature* σ^* *for the new claim sequence* $C^* := C \| (\mathsf{pk}, m)$*. Note that if* $C = \bot$ *and* $\sigma = \lambda$*, it signals that a new signature for* m *is to be created from scratch.*

- *The verification algorithm* $\mathsf{Vfy}(C, \sigma)$ *receives a claim sequence* C *and a signature* σ *as input. It outputs* 1 *if the signature* σ *is valid for* C *and* 0 *otherwise. Like for digital signature schemes (see Definition 2.2.4) we again allow the* Vfy *algorithm to use randomness, but expect its output given a fixed input to be deterministic.*

We require correctness as follows.

Definition 3.3.2 (Correctness of Sequential Aggregate Signature Schemes)**.** *A sequential aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{AggSign}, \mathsf{Vfy})$ *is* correct*, if for all security parameters* $\kappa \in \mathbb{N}$*, all claim sequences* C *consisting of claims* $c_i = (\mathsf{pk}_i, m_i)$ *where* pk_i *was output by* $\mathsf{Gen}(1^\kappa)$*, i.e. by computing* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\kappa)$*, and all signatures* σ *that were created by correct applications of the* AggSign *algorithm to create an aggregated signature on all claims of* C*, it holds that* $\mathsf{Vfy}(C, \sigma) = 1$*.*

Lysyanskaya et al. [Lys+04] define security for sequential aggregate signature schemes as follows. See Figure 3.2 for a visualization of the security experiment.

Definition 3.3.3 (SAS-EUF-CMA Security Experiment)**.** *The* SAS-EUF-CMA security experiment[4] *between an attacker* $\mathcal{A}$*, a challenger* $\mathcal{C}$ *and a sequential aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{AggSign}, \mathsf{Vfy})$ *consists of three phases as follows:*

Setup Phase. *The challenger* $\mathcal{C}$ *generates a key pair* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ *and gives the public key* pk *to the attacker* $\mathcal{A}$*.*

Query Phase. *The attacker* $\mathcal{A}$ *may request signatures for messages of his choosing from the challenger. If* $\mathcal{A}$ *sends a claim sequence* C_i*, a corresponding signature* σ_i *and a message* m_i *(*$i \in \mathbb{N}$*) to the challenger, he responds by computing* $\sigma_{\mathsf{Agg},i} \leftarrow \mathsf{AggSign}(\mathsf{sk}, C, \sigma_i, m_i)$ *and sending* σ_{Agg} *to* $\mathcal{A}$*. This step may be repeated at will by* $\mathcal{A}$*.*

Forgery Phase. *At the end of the experiment,* $\mathcal{A}$ *sends a tuple* (C^*, σ^*) *consisting of a claim sequence* C^* *and a signature* σ^* *to the challenger. Let* $q \in \mathbb{N}$ *be the number of messages* $\mathcal{A}$ *queried a signature for in the Query Phase.* $\mathcal{A}$ *is* successful*, if* $\mathsf{Vfy}(C^*, \sigma^*) = 1$ *and* C^* *is* non-trivial*, meaning that* C^* *contains at least one claim* $C^*[j] = (\mathsf{pk}, m^*)$ *using* pk *as its public key, such that for all* $i \in [q]$ *it holds that*

$$C_i \| (\mathsf{pk}, m_i) \neq (C^*[1], \ldots, C^*[j]),$$

i.e. $\mathcal{A}$ *has not called the* AggSign *oracle on the prefix of* C^* *up to the supposed forgery* (pk, m^*)*.*

Lysyanskaya et al. [Lys+04] also require for each signing query and in the forgery that no public key may be used more than once. As discussed in Section 3.2 (see the remark below Definition 3.2.3), we interpret such restrictions as a restriction of the scheme in question, rather then as a general rule. We have therefore removed this restriction from their security definition. Furthermore, Bellare, Namprempre, and Neven [BNN07] show that this restriction can be dropped and the scheme of Lysyanskaya et al. [Lys+04] still remains secure.

[4] The SAS in SAS-EUF-CMA stands for "Sequential Aggregate Signature".

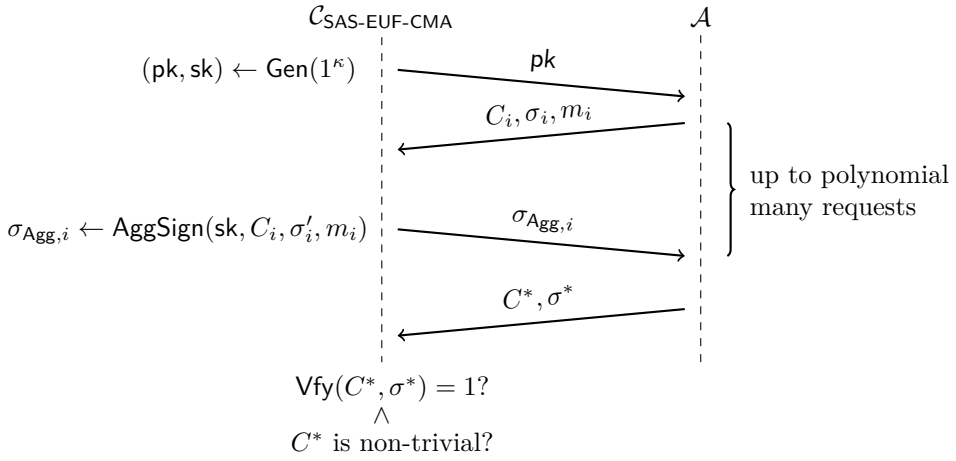

Figure 3.2: The SAS-EUF-CMA security experiment.

Observe that this security experiment, in contrast to other security experiments so far, allows the attacker $\mathcal{A}$ to reuse a message m_i in its forgery, that he requested a signature on during the Query Phase. The only restriction is that $\mathcal{A}$ never requested a sequential aggregate signature for a claim sequence that is equal to the "prefix" claim sequence up to (pk, m^*) in C^*. This way, for example, even a signature for a "simple" reordering of one of the queried claim sequences is seen as a successful forgery.

Not all publications on sequential aggregate signatures define security in this way. For example, Lu et al. [Lu+06] define the success requirement more like in the definition of AS-EUF-CMA by [Bon+03], i.e. attackers may not reuse messages that they already queried a signature for in the Query Phase. To be precise, in their definition, the attacker $\mathcal{A}$ only wins if there is a claim (pk, m^*) in C^* such that $\mathcal{A}$ never asked for a signature on m^* during the Query Phase. Observe that the security definition of [Lys+04] is stronger than and implies the one of [Lu+06].

This also implies that reorderings of the sequence, changes to the prefix and so on that are seen as successful forgeries in the model of [Lys+04] are *not* interpreted as successful in the definition of [Lu+06].

For the rest of this thesis, we use the unforgeability definition of [Lys+04] for sequential schemes and explicitly mention if results are dependent on the used security definition.

Definition 3.3.4 (SAS-EUF-CMA Security for Sequential Aggregate Signature Schemes)**.** *A sequential aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{AggSign}, \mathsf{Vfy})$ *is* SAS-EUF-CMA secure*, if all PPT algorithms* $\mathcal{A}$ *only have negligible success probability in the* SAS-EUF-CMA *security experiment, meaning it holds that*

$$\Pr\left[\mathcal{A}^{\mathcal{C}_{\mathsf{SAS\text{-}EUF\text{-}CMA}}}(\mathsf{pk}) = (C^*, \sigma^*) : \begin{array}{c}\mathsf{Vfy}(C^*, \sigma^*) = 1 \\ \wedge\, C^* \textit{ is non-trivial}\end{array}\right] \leq \mathsf{negl}(\kappa)$$

for a function negl*, which is negligible in the security parameter* κ*.*

3.3.1 Overview of Known Sequential Schemes

Several constructions for sequential aggregate signature schemes have been proposed. The first scheme was presented by Lysyanskava, Micali, Reyzin and Shacham [Lys+04]. They propose a construction based on trapdoor permutations in the random oracle model, where no public key can be used more than once per aggregate signature. Bellare, Namprempre, and Neven [BNN07] improve on the results of [Lys+04] by showing that this restriction on the use of public keys can safely be dropped from the scheme.

In [Lu+06] Lu et al. propose the first sequential aggregate scheme without random oracles, which is based on bilinear groups and the Computational Diffie-Hellman assumption. They introduce and employ a "certified-key model" in which the attacker has to prove that the public keys used in the forgery and in the signature queries were correctly generated, for example by sending the corresponding secret keys or using non-interactive zero knowledge proofs.

In [Sch11] Schröder gives a sequential aggregate signature scheme based on the signature scheme of [CL04], which has very short public keys (only two elements), but is also only secure in the certified-key model and relies on an interactive computational assumption.

[LLY13b; LLY13a] give two sequential schemes with short public keys of only a constant number of group elements. Their schemes are secure in the standard model, use pairings and are based on various non-interactive standard assumptions in the certified-key model.

Neven [Nev08] generalizes sequential aggregate signatures to the concept of *sequential aggregated signed data*. Here, the AggSign and Vfy algorithms no longer receive the public keys and messages as input, but only the current message and secret key (for the AggSign algorithm) and the aggregate so far (for both algorithms), which results in even bigger improvements in efficiency an bandwidth use. Neven gives a construction based on claw-free trapdoor permutations in the random oracle model. Unfortunately, as in many sequential aggregate signatures schemes ([Lys+04; Nev08; LLY13a; LLY13b], for example), the aggregation algorithm needs to verify the aggregate so far before further aggregation. Brogle, Goldberg, and Reyzin [BGR14] (originally published as an extended abstract [BGR12]) give a scheme in the random oracle model that does not need this verification step in the aggregation algorithm (they call this *lazy verification*) and also offers the same advantages that messages and public keys do not need to be passed to the AggSign algorithm as input, but has the disadvantage that signatures slightly grow in size. Concurrently, Fischlin, Lehmann, and Schröder [FLS12] also discuss the idea that the AggSign algorithm does not receive the public keys and messages so far. Their scheme is based on the scheme of [Bon+03], is secure in the random oracle model and signature do not grow in size. Bansarkhani, Mohamed, and Petzoldt [BMP16] give a sequential aggregate signature based on multivariate quadratic polynomials.

3.4 Synchronized Aggregate Signatures

To bridge the gap between fully flexible aggregation, which so far seems hard to achieve in the standard model, and the much more restricted sequential schemes, another form of restricted aggregation called *synchronized aggregation*

was proposed by Gentry and Ramzan [GR06]. Although the name suggests that signing and aggregation needs to be synchronized and done at the same time, this is not the case. Instead, the signing algorithm additionally takes a "time stamp" or "time period" in its input, which can usually be any integer or bit string and does not necessarily have to represent a real point in time.

All signatures that were created using the same time stamp can be aggregated in any order, no matter their actual time of creation and aggregation is a public operation, but signatures that were computed using different time stamps cannot be aggregated at all. So, for a fixed time stamp, aggregation is fully flexible. Unfortunately, in all known efficient synchronized schemes, signers may only create one signature per period.

The requirement that time stamps must be used is quite natural for many applications, but adds additional communication and planning overhead, since the signers must agree on some signing schedule. However, many applications like secure logging or collection of sensor data use regular reporting periods anyway. For other applications were such synchronization is not possible or feasible, sequential aggregation might actually be the more flexible approach, although aggregation itself is more restricted compared to synchronized schemes.

All currently known synchronized schemes also use a Setup algorithm, which takes as input the security parameter 1^κ and computes a set of public parameters, which every party needs to be able to compute its keys and signatures. It is assumed that this algorithm is honestly executed by a trusted party beforehand and all parties know the public parameters. In security experiments, the Setup algorithm is executed by the challenger.

Definition 3.4.1 (Synchronized Aggregate Signature Scheme)**.** *A* synchronized aggregate signature scheme *is a tuple* $\Sigma = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *of five PPT algorithms as follows:*

- *The setup algorithm* $\mathsf{Setup}(1^\kappa)$ *receives the security parameter* κ *as its input and outputs a set of public parameters* pp*.*
- *The key generation algorithm* $\mathsf{Gen}(1^\kappa, \mathsf{pp})$ *receives the security parameter* κ *and public parameters* pp *as its input and outputs a tuple* $(\mathsf{pk}, \mathsf{sk})$ *consisting of the* public key pk *and the* secret key sk*.*
- *The signature generation algorithm* $\mathsf{Sign}(\mathsf{sk}, m, s)$ *receives the secret key* sk*, a message* m *and a time stamp* s *as its input and outputs a signature* σ*.*
- *The aggregation algorithm* $\mathsf{Agg}(C_1, C_2, \sigma_1, \sigma_2)$ *takes as input two claim sequences* C_1 *and* C_2*, as well as corresponding signatures* σ_1 *and* σ_2 *and outputs an aggregate signature* σ_{Agg} *for all claims in* C_1 *and* C_2 *or* $\perp$*, if the signatures* σ_1 *and* σ_2 *were not created using the same time stamp.*
- *The verification algorithm* $\mathsf{Vfy}(C, \sigma)$ *receives a claim sequence* C *and a signature* σ *as input. It outputs* 1 *if the signature* σ *is valid for* C *and* 0 *otherwise. The* Vfy *algorithm may use randomness, but its output given a fixed input must be deterministic.*

We now state the security definition for synchronized aggregate signature schemes by [AGH10], which is also based on the security definition of [Bon+03] for fully flexible aggregate signature schemes. See Figure 3.3 for a visualization of the security experiment.

Definition 3.4.2 (SyncAS-EUF-CMASecurity Experiment)**.** *The* SyncAS-EUF-CMA security experiment[5] *between an attacker $\mathcal{A}$, a challenger $\mathcal{C}$ and a synchronized aggregate signature scheme $\Sigma = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ consists of three phases as follows:*

Setup Phase. *The attacker $\mathcal{A}$ sends a number $n \in \mathbb{N}$ of its choosing to the challenger $\mathcal{C}$. The challenger then runs $\mathsf{Setup}(1^\kappa)$ to obtain the public parameters pp, generates n key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\kappa, \mathsf{pp})$ and gives $(\mathsf{pp}, \mathsf{pk}_1, (\mathsf{pk}_2, \mathsf{sk}_2), \ldots, (\mathsf{pk}_n, \mathsf{sk}_n))$ to the attacker $\mathcal{A}$.*

Query Phase. *For each time period, the attacker $\mathcal{A}$ may request* one *signature valid under pk_1 for a message of his choosing from the challenger. If $\mathcal{A}$ sends a message m_i ($i \in \mathbb{N}$) in time period s_i to the challenger, he responds by computing $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_1, m_i, s_i)$ and sending σ_i to $\mathcal{A}$.*

Forgery Phase. *At the end of the experiment, $\mathcal{A}$ sends a tuple (C^*, σ^*) consisting of a claim sequence $C^* = ((\mathsf{pk}_1, m_1^*), \ldots, (\mathsf{pk}_n, m_n^*))$ and a signature σ^* to the challenger $\mathcal{C}$. Note that the public keys in C^* must be equal to the public keys given to $\mathcal{A}$ by the challenger. Let m_i be the message $\mathcal{A}$ queried a signature for in time period s_i and let $q \in \mathbb{N}$ be the number of time periods used. $\mathcal{A}$ is* successful*, if*

$$\mathsf{Vfy}(C^*, \sigma^*) = 1 \quad \textit{and} \quad m_1^* \notin \{m_1, \ldots, m_q\}.$$

The security experiment has several restrictions:

1. The attacker is only successful if he never asks for a signature on the message m_1^* in *any* time period. A seemingly stronger security notion might not restrict the attacker in this way, since signatures of different time periods are not "compatible" to one another. However, as [AGH10] already note, this stronger notion can be achieved by any scheme that satisfies the above notion by incorporating the current time period into each message during signing.

2. All non-challenge public keys are honestly chosen by the challenger, in contrast to the the definitions for fully and sequentially aggregate signature schemes, where the attacker has full control over these keys and may chose them maliciously. Alternatively, a Knowledge of Secret Key model [Bar+04; AGH10] could also be used, where the attacker needs to prove that he knows the corresponding secret key to the public key that he wants to use. This restriction is necessary for the security of all known synchronized schemes [AGH10; HW18], except for the identity-based scheme of [GR06].

3. The attacker may only query one signature per time period. This restriction is necessary for the security of all known synchronized schemes [GR06; AGH10; HW18], but might not be necessary in general. In fact, [GR06; AGH10] also propose variants of their schemes in which parties can sign a fixed number of messages per time period. However, these variants are less efficient, since their public keys grow linearly in the number of allowed messages per time period: if each signer should be able to sign n messages, they basically create n different key pairs.

[5] The SyncAS in SyncAS-EUF-CMA stands for "Synchronized Aggregate Signature".

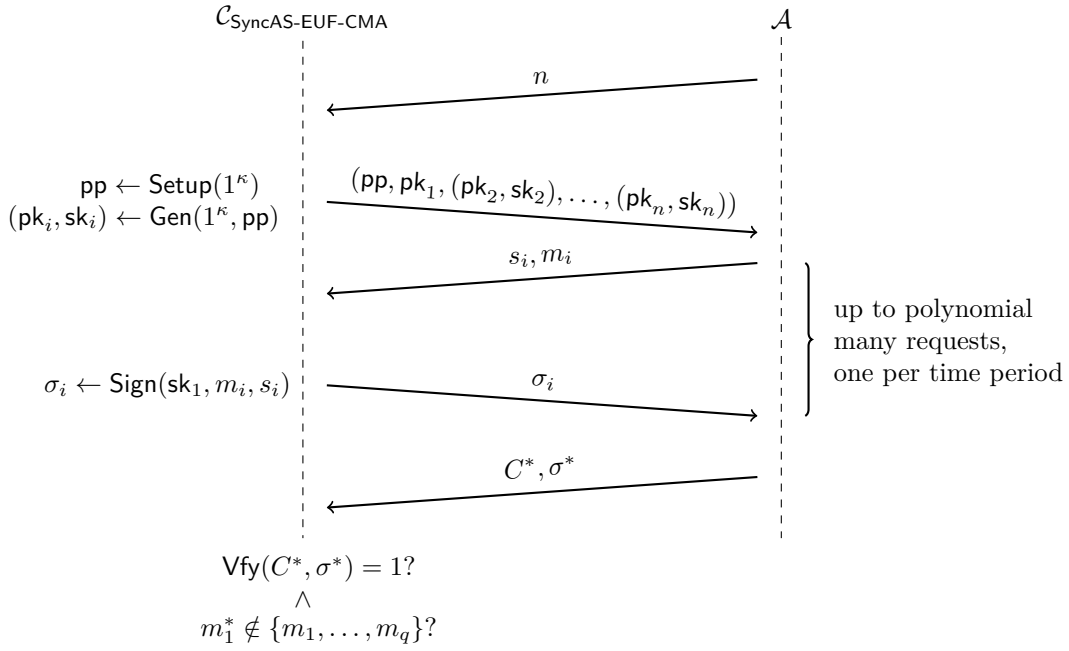

Figure 3.3: The SyncAS-EUF-CMA security experiment.

Definition 3.4.3 (SyncAS-EUF-CMA Security for Synchronized Aggregate Signature Schemes)**.** *A synchronized aggregate signature scheme* $\Sigma = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *is* SyncAS-EUF-CMA secure*, if all PPT algorithms* $\mathcal{A}$ *only have negligible success probability in the* SyncAS-EUF-CMA *security experiment, meaning it holds that*

$$\Pr[\mathcal{A}^{\mathcal{C}_{\mathsf{SyncAS\text{-}EUF\text{-}CMA}}}(\mathsf{pp}, \mathsf{pk}_1, (\mathsf{pk}_2, \mathsf{sk}_2), \ldots, (\mathsf{pk}_n, \mathsf{sk}_n)) = (C^*, \sigma^*) : \mathsf{Vfy}(C^*, \sigma^*) = 1 \wedge m_1^* \notin \{m_1, \ldots, m_q\}] \leq \mathsf{negl}(\kappa)$$

for a function negl*, which is negligible in the security parameter* κ*.*

3.4.1 Overview of Known Synchronized Schemes

At the time of publication of this thesis and to the best of our knowledge, only three synchronized aggregate signature schemes are known.

Gentry and Ramzan [GR06] initially proposed the idea of synchronized aggregation. They gave an identity-based scheme based on the Computational Diffie-Hellman assumption in the random oracle model, where only signatures can be aggregated that share a specific random value.

Ahn, Green, and Hohenberger [AGH10] then generalized the concept and construct a synchronized aggregate signature scheme using bilinear groups secure in the standard model under the Computational Diffie-Hellman assumption. They also give a more efficient variant of their scheme in the random oracle model. See Section 5.3.7 and Section 5.3.8 for detailed discussions of their schemes in the context of deaggregation security.

In 2018, Hohenberger and Waters [HW18] proposed a scheme based on the RSA assumption in the standard model. Their scheme is also the first synchronized aggregate signature scheme which does not use bilinear groups, but has the additional restriction that it only works for a previously fixed bounded number of time periods.

3.5 Identity-Based Aggregate Signatures

Additionally to the regular types of aggregate signatures schemes discussed above, there also exist several schemes that follow the well known approach of identity-based cryptography [Sha84]. In an identity-based scheme, instead of public keys, some form of unique and easy to remember identifier, like the e-mail address of the signer, is used to verify signatures, i.e. this *identity* has the same function as a normal public key, but does not need to be computed by a Gen algorithm.

This approach offers the advantage that public keys do not need to be securely stored in a public key infrastructure and users can easily get a hold of the necessary information to verify signatures or to encrypt data in the case of identity-based encryption.

However, a trusted authority is needed that uses a *master secret key* to compute fitting secret keys corresponding to these identifiers, so identity-based schemes are only suitable for applications where such a trusted authority can be established, for example within a company network. Identity-based constructions can be found in the following publications:

Fully Flexible Aggregation: [XZF05; CLW05; Che+06] give several constructions of identity-based aggregate signatures in the random oracle model using bilinear maps.

Bagherzandi and Jarecki [BJ10] give an identity-based aggregate signature scheme based on the RSA assumption in the random oracle model. Their scheme needs to use two rounds of communication between the signers before they can sign messages and aggregate signatures.

Hohenberger, Sahai, and Waters [HSW13] give a construction using the graded multilinear maps of Garg, Gentry, and Halevi [GGH12].

Sequential Aggregation: Boldyreva et al. [Bol+07] introduce the concept of identity-based sequential aggregate signature schemes and give a construction which is secure in the random oracle model. Their scheme is based on a novel interactive complexity assumption, which was subsequently shown to not hold by Hwang, Lee, and Yung [HLY09]. They also prove that the scheme is not secure and in fact universally forgeable.

Dou et al. [Dou+09] propose a non-interactive and an interactive identity-based sequential aggregate signature based on the RSA assumption (in the interactive version, signers first need to exchange random values before they can sign and aggregate).

Tsai, Lo, and Wu [TLW13] however show that the non-interactive scheme of [Dou+09] is not secure and propose a scheme based on the RSA assumption.

Synchronized Aggregation: Gentry and Ramzan [GR06] introduce the concept of synchronized aggregation and give an identity-based construction using bilinear maps which is secure in the random oracle model under the Computational Diffie-Hellman assumption. A synchronized identity-based scheme based on the RSA assumption is given by Hohenberger and Waters [HW18].

Partial Aggregation: Identity-based aggregate signatures with a restricted form of aggregation, called partial aggregation, are discussed in [Her06; Sel+12]. Here, only some parts of the signatures are aggregated and other parts (usually random strings) need to be propagated, i.e. the signatures grow linearly in size in the number of claims.

We do not discuss the identity-based setting further in this thesis, but the results presented here (especially the results for fault-tolerance) should be transferable to this setting as well.

Chapter 4

Fault-Tolerance

4.1 Introduction

In this chapter, we discuss the first main contribution of this thesis, which is *fault-tolerance* of aggregate signatures. One problem that all known aggregate signature schemes so far have in common is that they cannot tolerate the aggregation of invalid signatures. If even one single invalid signature gets aggregated to an already aggregated signature σ_{Agg}, the complete aggregate will become invalid. This is also the case if a "wrong" message is included for verification, i.e. a claim that is not valid for the aggregate signature was added to the claim sequence. In both cases, the verification algorithm can give no information about which message-signature pair is the reason for the failure and if other message-signature pairs are valid. The aggregate signature can therefore no longer be used to convince a verifier of the authenticity and integrity of the previously signed messages, since the Vfy algorithm will output 0, even though all other messages might have been correctly signed before the error occurred. This essentially renders the aggregate signature useless after an invalid signature is added.

This is very undesirable, because it would be necessary to re-sign all data after an error like this occurred, which is cost-intensive and could also lead to security problems, since it can no longer be securely inferred which messages were actually signed beforehand. In some applications it might even be impossible to re-sign the data, especially if multiple parties are involved in the computation of the signatures.

The inclusion of invalid signatures can not only occur by adversarial behavior, but also because of simple programming errors, system failures, storage problems and so on and is therefore not only a concern from a security perspective, but also from an availability perspective. One single bit flip in the message or signature is enough to unintentionally turn an individual signature invalid (if this would not be the case, it would be easy to forge a signature) and therefore render σ_{Agg} useless. For example, let $C \coloneqq (c_1 \coloneqq (\mathsf{pk}_1 = g^{x_1}, m_1), c_2 \coloneqq (\mathsf{pk}_2 \coloneqq g^{x_2}, m_2))$ be a claim sequence for the BGLS aggregate signature scheme (for a brief overview, see Section 3.2.1) with a valid aggregate signature σ_{Agg}. Since σ_{Agg} is valid, it must be of the form

$$\sigma_{\mathsf{Agg}} = H(m_1)^{x_1} \cdot H(m_2)^{x_2}.$$

Let $c_3 := (\mathsf{pk}_3 := g^{x_3}, m_3)$ be a third claim and suppose that an invalid signature $\sigma_{\mathsf{invalid}} \neq H(m_3)^{x_3}$ is aggregated to σ_{Agg} by computing $\sigma'_{\mathsf{Agg}} := \sigma_{\mathsf{Agg}} \cdot \sigma_{\mathsf{invalid}}$. Then σ'_{Agg} will be invalid for the claim sequence $C' = (c_1, c_2, c_3)$, because

$$\begin{aligned} e(\sigma'_{\mathsf{Agg}}, g) &= e(H(m_1)^{x_1} \cdot H(m_2)^{x_2} \cdot \sigma_{\mathsf{invalid}}, g) \\ &= e(H(m_1), g^{x_1}) \cdot e(H(m_2), g^{x_2}) \cdot e(\sigma_{\mathsf{invalid}}, g) \\ &\neq e(H(m_1), g^{x_1}) \cdot e(H(m_2), g^{x_2}) \cdot e(H(m_3)^{x_3}, g) \\ &= \prod_{i=1}^{3} e(H(m_i), \mathsf{pk}_i) \end{aligned}$$

and the Vfy algorithm would therefore return 0. Although σ_{Agg} was initially correctly computed, all information it stored is now lost.

A seemingly natural approach to solve this problem seems to be to verify each signature before aggregating it, as is actually done in some (but not all) schemes ([Lys+04; Nev08; LLY13a; LLY13b], for example). While this is generally undesirable, because it adds an additional and potentially big unwanted computational overhead to each aggregation step[1], it seems like this might already provide fault-tolerance. Unfortunately, this is not the case, since such faults can also happen *after* the signatures were computed and aggregated, even if all steps were done correctly. If a message gets changed for any reason (for example by some error of the storage medium), this will also mean that the already aggregated signature for this message becomes faulty, since it will not be valid for the modified message. Therefore, the complete aggregate signature becomes invalid in this case as well.

For standard digital signatures this behavior is actually advantageous and desired, since it ensures the integrity of each stored message. However, for aggregate signatures, a potentially large set of messages can be influenced. For example, a secure logging scheme that signs its entries by using an aggregate signature might store thousands, if not millions of log entries. If the complete aggregate becomes invalid and unusable, no information whatsoever about which message was changed can be inferred from the now invalid aggregate. The stored aggregate signature therefore is a single point of failure: If it becomes invalid, the application using it looses all security provided by the signature.

To solve this problem, we introduce the concept of fault-tolerant aggregate signature schemes, which are able to tolerate the inclusion of a specific number of invalid (or faulty) signatures. In such a scheme, the verification algorithm does not output boolean values like 1 for "valid" and 0 for "invalid" but instead outputs a list of validly signed messages and will leave out all messages that are not validly signed.

[1] Verifying signatures is almost always a computationally expensive operation. For example, several pairings need to be evaluated in the Vfy algorithm of the BGLS scheme. However, if such a verification step is present in the Agg algorithm of a scheme, it cannot necessarily be removed after adding fault-tolerance to the scheme, since its unforgeability security proof might critically rely on this step. For example, the schemes of [Lys+04; Nev08] become breakable if this step is removed [BGR14]. So, before removing this step, one needs to check the unforgeability security proof of the scheme carefully to see if it would still go through after its removal.

4.1.1 Contribution

We provide a formal framework of definitions for fault-tolerant aggregate signatures and present a generic black-box construction which can be used to turn *any* aggregate signature scheme into a fault-tolerant one. This construction is also the *first* construction of a fault-tolerant scheme. It combines a standard aggregate signature scheme with a cover-free family [KS64] to provide fault-tolerance and has a tight security reduction to the underlying signature scheme. The basic construction has the shortcoming that only a fixed number of messages can be aggregated, but we also show how to achieve unbounded aggregation.

We also investigate the relationship between fault-tolerance and signature size and show that signatures of fault-tolerant aggregate signature schemes necessarily need to grow in size.

Furthermore, we also adapt the formal framework to the case of fault-tolerant sequential aggregate signatures. Here, slightly different definitions are needed, since no individual signatures exists and the concept of a "faulty signature" needs to be defined differently. However, the construction to achieve this different definition is essentially the same as the construction for aggregate signatures.

For concreteness, we explicitly describe how to instantiate our scheme with a cover-free family based on polynomials over a finite field [KRS99], which has a compact representation. This leads to an instantiation featuring short aggregate signatures relative to the number n of individual signatures that are aggregated (provided that the maximal number of faults the scheme should tolerate is relatively small compared to n). To be more precise, signatures only grow logarithmically in the number of aggregated messages. Our scheme therefore achieves the *optimal compression ratio* for fault-tolerant aggregate signature schemes.

As an additional feature, our construction allows the verification of individual claims in a fashion that is more efficient than verifying the complete aggregate. This provides an additional level of flexibility to the signature scheme as demanded by certain applications such as secure logging [MT09]. Our construction has two restrictions:

- We need to *assume* that aggregates may only contain invalid individual signatures up to a previously *fixed* upper bound d, which is smaller than the actual number of signed messages. If for some reason this bound is exceeded, the faulty signatures may influence the verifiability of other messages, as is the case for common aggregate signatures. This is also comparable to error-correction codes (which are related to cover-free families), where only a specific number of errors can be located. In our scheme, the concrete relationship between the upper bound d and the total amount n of messages is dependent on the used cover-free family.

- Our scheme only supports a slightly restricted form of aggregation, which is not fully flexible, but more flexible than sequential and synchronized aggregation. We discuss this further in Section 4.5.

Remark. The main contents of this chapter are taken almost entirely from [Har+16], which was published at Public Key Cryptography 2016, and [Har+17a], which was published at Provable Security 2017, and the corresponding full version [Har+17b]. Significant parts of these publications are reproduced here without or with only minor modifications, and without specific designation.

4.1.2 Related Work

Independently from us, Idalino et al. [Ida+15] and Idalino [Ida15] also developed concepts (*modification location signature schemes* and *level-d signature aggregation*) very similar to fault-tolerance and give constructions based on cover-free families to achieve them that are also similar to our construction of fault-tolerance.

Idalino and Moura [IM18] improve the efficiency of our unbounded construction by using nested cover-free families and offer a better compression ratio than our unbounded construction using a monotone cover-free family, where signatures unfortunately grow linearly in size.

Apart from the mentioned publications and to the best of our knowledge, no other work directly related to fault-tolerance exists. A general overview over publications related to aggregate signatures can be found in the overview sections of Chapter 3.

4.1.3 Overview

In Section 4.2 we first give an overview of the basic idea of our construction to achieve fault-tolerance. Section 4.3 then gives a formal framework for fault-tolerance by formally defining it and all necessary prerequisites. Section 4.4 discusses the connection between signature size and fault-tolerance and in Section 4.5 we present our black-box construction using cover-free families that can be used to turn any aggregate signature scheme into a fault-tolerant one. We also discuss how to generalize this construction to allow unbounded aggregation in Section 4.5.1 and discuss the additional selective verification feature in Section 4.5.2. In Section 4.5.3 we then show an instantiation using a concrete cover-free family based on polynomials over a finite field. Section 4.6 then discusses how to apply and translate these concepts to the case of sequential aggregate signature schemes. To conclude, Section 4.7 shows how fault-tolerance can be applied to construct a secure and robust logging scheme.

4.2 Basic Idea of Our Construction

To get a glimpse of our generic construction of a fault-tolerant aggregate signature scheme, we now informally illustrate the basic idea.

Let an ordinary aggregate signature scheme (e.g., the BGLS scheme) and n individual signatures $\sigma_1, \ldots, \sigma_n$ generated using this scheme be given. Our goal for this example is to tolerate $d = 1$ faulty individual signatures. To achieve this, our approach is to choose m subsets

$$T_1, \ldots, T_m \subset \{\sigma_1, \ldots, \sigma_n\}$$

of individual signatures and aggregate the signatures of each subset, thereby yielding aggregate signatures $\tau_1, \ldots, \tau_m$, such that

1. m is (significantly) smaller than n and
2. even if one of the individual signatures is faulty and the corresponding aggregate signatures τ_i will be invalid, all other individual signatures σ_j are aggregated into at least one different, valid signature τ_k, that can then be used to deduce the validity of σ_j.

For example, consider the following binary 4×6 matrix

$$A := (a_{i,j}) := \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

for which $n = 6$ and $m = 4$.

A describes a solution to the above mentioned problem as follows: The 1-entries in column j indicate in which T_i the individual signature σ_j is contained. Consequently, the 1-entries in row i indicate the σ_j contained in T_i. More precisely, $T_i := \{\sigma_j \colon a_{i,j} = 1\}$ and τ_i is the aggregate of all $\sigma_j \in T_i$.

For the matrix above, an aggregate signature $\tau = (\tau_1, \tau_2, \tau_3, \tau_4)$ for individual signatures $\sigma_1, \ldots, \sigma_6$ would therefore be computed in the following manner:

$$\tau = \begin{pmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \end{pmatrix} = \begin{pmatrix} \mathsf{Agg}(\sigma_1, \sigma_4, \sigma_6) \\ \mathsf{Agg}(\sigma_1, \sigma_2, \sigma_5) \\ \mathsf{Agg}(\sigma_2, \sigma_3, \sigma_4) \\ \mathsf{Agg}(\sigma_3, \sigma_5, \sigma_6) \end{pmatrix} \hat{=} \begin{pmatrix} \sigma_1 & & & \sigma_4 & & \sigma_6 \\ \sigma_1 & \sigma_2 & & & \sigma_5 & \\ & \sigma_2 & \sigma_3 & \sigma_4 & & \\ & & \sigma_3 & & \sigma_5 & \sigma_6 \end{pmatrix},$$

where Agg informally denotes the aggregation algorithm of the underlying aggregate signature scheme.

Let us assume that only one signature σ_j is faulty. Then all τ_i are faulty where $a_{i,j} = 1$. However, because all other σ_k were also aggregated into at least one different τ_i, we can still derive the validity of σ_k.

For a concrete example, suppose σ_1 is faulty. Then τ_1 and τ_2 will be faulty, whereas τ_3 and τ_4 remain valid. We see that σ_2, σ_3 and σ_4 occur in τ_3, and σ_5, σ_6 occur in τ_4. So we can still infer that m_2, m_3, m_4, m_5 and m_6 were validly signed from τ_3 and τ_4, although the verification of τ_1 and τ_2 will fail, which in turn implies that m_1 was *not* validly signed.

The matrix A defined above has the property that it can tolerate one faulty signature, i.e. if just one signature is faulty, then all other messages can still be verified. This is not possible if two or more faulty signatures are aggregated. Suppose that σ_1 and σ_2 are faulty. In this case, τ_1, τ_2 and τ_3 become invalid and τ_4 is the only valid signature. We could still derive the validity of σ_3, σ_5 and σ_6, because τ_4 is valid. However, the validity of σ_4 can no longer be verified, since it was never aggregated to τ_4. So, although σ_4 was valid when it was aggregated, the aggregation of only two invalid signatures now implies that we can no longer derive any information about its validity. For an even stronger example, suppose σ_1 and σ_3 would be invalid – then all verification steps will fail and we cannot deduce the validity of any message.

Our construction basically works as described in the example above. Note that therefore our scheme does not support fully flexible aggregation: Each column of A can only be used to hold one individual signature, so two aggregate signatures where the same column is used cannot be aggregated further without losing the guarantee of fault-tolerance. However, this only slightly restricts aggregation: Individual signatures can always be aggregated and two aggregate signatures can be aggregated if no column is used in both. As long as this requirement is met, signatures can be aggregated in any order. Signatures of our scheme will only be valid for one specific claim sequence, since we use the

position of each claim in the sequence to decide which column of the matrix was used to aggregate its individual signature. Still, the order of aggregation can be arbitrary. Signers simply need to agree on the "position" of their claim in the sequence and adhere to it when aggregating. This notion is sufficient for many use cases, we discuss this further in Section 4.5.

The construction of matrices that can tolerate $d > 1$ faulty signatures is more intricate, but incidence matrices belonging to d-cover-free families imply the desired property. Informally speaking, in such a matrix the "superposition" $\vec{s}$ of up to d arbitrary column vectors $\vec{a_1}, \ldots, \vec{a_d}$, i.e., the vector $\vec{s}$ which has a 1 at position ℓ if at least one of the vectors $\vec{a_1}, \ldots, \vec{a_d}$ has a 1 at this position, does not "cover" any other distinct column vector $\vec{a_j}$ ($j \notin \{1, \ldots, d\}$). In other words, there is at least one position ℓ such that $\vec{a_j}$ has a 1 at this position but $\vec{s}$ shows a 0. This implies that if at most d individual signatures (each belonging to one column) are invalid, then each distinct individual signature is contained in at least one valid aggregate signature, and the corresponding message can therefore be securely verified. Hence, applying such a matrix, as sketched above, implies that any subset of faulty individual signatures of size up to d will not compromise the trustworthiness of any other message.

4.3 Framework for Fault-Tolerance

Before we can introduce the formal definition of fault-tolerance for aggregate signature schemes, we first have to define and discuss a few preliminaries.

Remark. We focus on the case of aggregate signatures for the most part of this chapter. The results presented can however also be applied to synchronized aggregate signature schemes, with the usual restriction that only signatures created using the same time stamp can be aggregated. For sequential aggregate signature schemes, slightly different definitions are needed, as discussed in Section 4.6.

The intuitive difference between a fault-tolerant and an ordinary aggregate signature scheme is that its verification algorithm does not only output a boolean value 1 or 0, but instead outputs the set of valid claims. If the signature contains more errors than the scheme can cope with, Vfy may output just a subset of the valid claims. Other claims may be clearly false or just not certainly true, since the verification algorithm ought to be conservative and reject a claim in case of uncertainty.

We update the definition of aggregate signatures to reflect these changes by defining *aggregate signature schemes with list verification*. The name "list verification" is chosen to indicate the changes in syntax, in particular that the verification algorithm outputs a multiset or list, instead of just 1 or 0.

In this chapter, σ will usually be used to denote signatures of standard aggregate signature schemes, whereas τ will mostly refer to signatures of aggregate signature schemes with list verification and fault-tolerant aggregate signature schemes. As usual, the aggregation algorithm is called with two claim sequences, hence, before aggregating, a single claim c must be converted to a claim sequence $C = (\perp, \ldots, \perp, c)$. We emphasize once more that for our scheme, the positions of the claims in the sequence are important, so this conversion of a single claim c to a claim sequence should also be used to assign a position to c.

Definition 4.3.1 (Aggregate Signature Scheme with List Verification)**.** *An* aggregate signature scheme with list verification *is a tuple* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *of four PPT algorithms as follows:*

- *The key generation algorithm* $\mathsf{Gen}(1^\kappa)$ *receives the security parameter* κ *as its input and outputs a tuple* $(\mathsf{pk}, \mathsf{sk})$ *consisting of the* public key pk *and the* secret key sk*.*
- *The signature generation algorithm* $\mathsf{Sign}(\mathsf{sk}, m)$ *receives the secret key* sk *and a message* m *as its input and outputs a signature* τ*.*
- *The aggregation algorithm* $\mathsf{Agg}(C_1, C_2, \tau_1, \tau_2)$ *takes as input two* exclusively mergeable *claim sequences* C_1 *and* C_2*, as well as corresponding signatures* τ_1 *and* τ_2 *and creates an aggregate signature* τ_{Agg}*, certifying the validity of the claim sequence* $C_1 \sqcup C_2$*.*
- *The verification algorithm* $\mathsf{Vfy}(C, \tau)$ *takes as input a claim sequence* C *and an aggregate signature* τ *for* C*. It outputs a multiset of claims* $C_{\text{valid}} \subseteq \mathsf{elem}(C)$ *specifying the valid claims in* τ*. Note that this may be a proper subset of* $\mathsf{elem}(C)$*, or even empty, if none of the claims can be derived from* τ *for certain. Again,* C *may contain* $\perp$ *as a claim placeholder.*

Σ *is required to be* correct *as defined in Definition 4.3.10 and the following paragraphs.*

Remark. Since we're almost exclusively interested in aggregate signature schemes with list verification for the rest of this chapter, we often leave out "with list verification" for brevity.

The security experiment and security definition for aggregate signatures with list verification is a direct adaption of the standard security experiment of [Bon+03] as also presented in Definition 3.2.3 and Definition 3.2.4. We present the definitions in the concrete form (see the end of Section 2.2.2 for a discussion of concrete security definitions) to be consistent with our original publication on fault-tolerance [Har+16].

Definition 4.3.2 (AS-EUF-CMA Security Experiment for Aggregate Signature Schemes with List Verification)**.** *The* AS-EUF-CMA *security experiment between an attacker* $\mathcal{A}$*, a challenger* $\mathcal{C}$ *and an aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *with list verification consists of three phases as follows:*

Setup Phase. *The challenger* $\mathcal{C}$ *generates a key pair* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ *and gives the public key* pk *to the attacker* $\mathcal{A}$*.*

Query Phase. *The attacker* $\mathcal{A}$ *may request signatures for messages of his choosing from the challenger. If* $\mathcal{A}$ *sends a message* m_i *(*$i \in \mathbb{N}$*) to the challenger, he responds by computing* $\tau_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$ *and sending* τ_i *to* $\mathcal{A}$*. This step may be repeated at will by* $\mathcal{A}$*.*

Forgery Phase. *At the end of the experiment,* $\mathcal{A}$ *sends a tuple* (C^*, τ^*) *consisting of a claim sequence* C^* *and a signature* τ^* *to the challenger* $\mathcal{C}$*. Let* $q \in \mathbb{N}$ *be the number of messages* $\mathcal{A}$ *queried a signature for in the Query Phase. Then* $\mathcal{A}$ *is* successful*, if*

$$\exists c^* = (\mathsf{pk}, m^*) \in C^* \;\; \textit{such that} \;\; c^* \in \mathsf{Vfy}(C^*, \tau^*) \;\; \textit{and} \;\; m^* \notin \{m_1, \ldots, m_q\}.$$

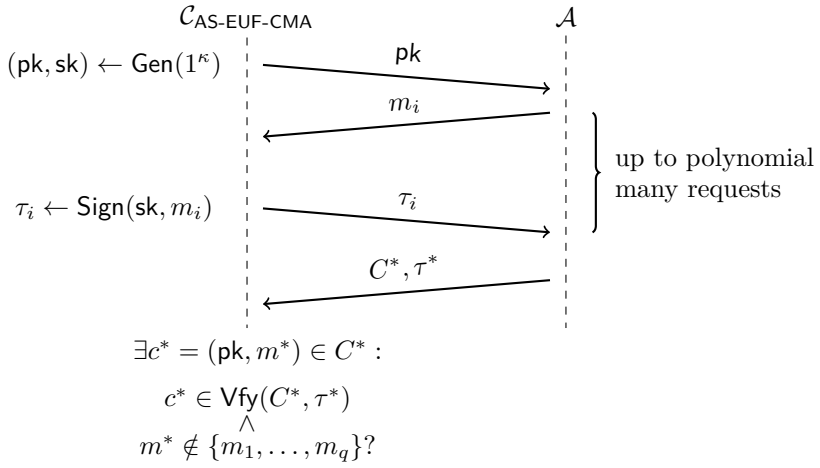

Figure 4.1: The AS-EUF-CMA security experiment for aggregate signature schemes with list verification.

Definition 4.3.3 (AS-EUF-CMA Security for Aggregate Signature Schemes with List Verification)**.** *An aggregate signature scheme with list verification is* (t, q, ε)-AS-EUF-CMA secure *if there is no attacker* $\mathcal{A}$ *running in time at most* t*, making at most* q *queries to the signature oracle and winning in the* AS-EUF-CMA *security experiment for schemes with list verification with probability at least* ε*.*

To be able to define fault-tolerance, we need to clearly define when signatures are to be interpreted as "faulty". For this, we propose the concept of *regular signatures*. Informally, a signature is regular if it is created by correctly running the algorithms of a signature scheme Σ.

Definition 4.3.4 (Regular Signatures)**.** *Let* C *be a claim sequence,* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *an aggregate signature scheme with list verification*[2] *and* τ *be a signature of* Σ*.* Regularity *is defined recursively as follows:*

- *If* $(\mathsf{pk}, \mathsf{sk})$ *is in the image of* $\mathsf{Gen}(1^\kappa)$ *and* $C = ((\mathsf{pk}, m))$ *for a message* m*, and if* τ *is in the image of* $\mathsf{Sign}(\mathsf{sk}, m)$*, then* τ *is said to be* regular for C *and for any claim sequence obtained by prepending any number of claim placeholders* $\perp$ *to* C*.*
- *If* τ_1 *is* regular *for a claim sequence* C_1*,* τ_2 *is regular for another claim sequence* C_2*, and* C_1, C_2 *are exclusively mergeable, then* τ *is* regular *for* $C_1 \sqcup C_2$ *if* τ *is in the image of* $\mathsf{Agg}(C_1, C_2, \tau_1, \tau_2)$*.*
- *The empty signature* λ *is* regular *for the claim sequences containing only* $\perp$ *and the empty claim sequence* $()$*. No other signature than* λ *is regular for these claim sequences.*

If a signature τ *is not regular for a claim sequence* C*, it is called* irregular *for* C*.*

[2] Regularity can easily be adapted to standard aggregate signatures without list verification, synchronized aggregate signatures and digital signatures, as well. For sequential aggregate signatures this concept needs be defined differently, see Section 4.6.

Definition 4.3.5 (Amount of Errors)**.** *Let $M = \{(c_1, \tau_1), \ldots, (c_n, \tau_n)\}$ be a multiset of claim and signature pairs, which is partitioned into two multisets M_{irreg} and M_{reg}, containing the pairs for which τ_i is irregular for $C = (c_i)$ and regular for C, respectively. Then the multiset M* contains *d* errors*, if $|M_{\text{irreg}}| = d$.*

We view irregular signatures as erroneous and all our fault-tolerance guarantees do only concern regular signatures. This way we obtain a very clear and precise definition of what should be seen as an "error" or as a "faulty signature". Furthermore, this way, the following definition of fault-tolerance can also be interpreted as a generalization of correctness. However, note that it is technically possible that for a given signature scheme there may be irregular signatures, which still are valid (i.e. the Vfy algorithm outputs 1, even though they are irregular). Although this might seem counter-intuitive at first, we argue that this definition nevertheless correctly captures the meaning of "faulty" for all practical cases.

If valid, but irregular, signatures exist for a given scheme, then in most cases it should be possible to reformulate the scheme so that these signatures are also regular. The Sign algorithm (and maybe also the Gen algorithm) simply needs to be redefined, so that these irregular signatures are also in the image of Sign and we can assume that all schemes are formulated in this way.

If such a redefinition is not (efficiently) possible, than this implies that it is difficult to compute these signatures even if the secret key is known. Therefore it is not possible for a PPT attacker to somehow advantageously use these signatures with more than negligible probability.

Example 4.3.6. Most schemes do not have valid but irregular signatures. For example, the schemes of [Bon+03; Lu+06; AGH10] have no such signatures (see the Lemmas 5.3.1, 5.3.28 and 5.3.37).

Still, it is easy to see that such schemes exist: suppose $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ is a digital signature scheme that has no irregular and valid signatures. We construct a scheme $\Sigma' = (\mathsf{Gen}, \mathsf{Sign}', \mathsf{Vfy}')$ from it that does have valid signatures that are irregular. The Sign' algorithm runs Sign and appends a 0-bit to the end of the computed signature, i.e. $\mathsf{Sign}'(\mathsf{sk}, m) = \mathsf{Sign}(\mathsf{sk}, m) \| 0$. The new Vfy' algorithm simply removes the last bit of the given signature and verifies it by running Vfy, i.e. it computes $\mathsf{Vfy}'(\mathsf{pk}, m, \sigma \| b) = \mathsf{Vfy}(\mathsf{pk}, m, \sigma)$. If Σ is $\mathsf{EUF\text{-}CMA}$ secure, then so is Σ', but now for every message there exists at least one irregular but valid signature, namely the signature output by Sign, but with an appended 1-bit. However, it would also be easy to fix this problem here by reformulating the Sign' algorithm, so that it appends a random bit $b \leftarrow \{0, 1\}$ or by simply removing this bit altogether.

Definition 4.3.7 (Fault-Tolerance for Aggregate Signature Schemes)**.** *An aggregate signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ with list verification is* tolerant against *d* errors*, if for any multiset $M = \{(c_1, \tau_1), \ldots, (c_n, \tau_n)\}$ of claim and signature pairs containing at most d errors, for any signature τ that was aggregated from the signatures in M in arbitrary order and the corresponding claim sequence C, which may additionally contain any number of claim placeholders $\perp$, we have*

$$R \subseteq \mathsf{Vfy}(C, \tau),$$

where R is the multiset of all the claims c_i of the pairs $(c_i, \tau_i) \in M_{\text{reg}}$. In other words, Vfy *outputs at least all claims of regular signatures.*

Intuitively, one would expect $R = \mathsf{Vfy}(C, \tau)$. Unfortunately, this is not achievable in general, as the aggregation of multiple irregular signatures might actually "cancel out" the irregularity and result in a regular and valid aggregate signature. See the following example using the BGLS aggregate signature scheme:

Example 4.3.8. Let $\sigma_1 = H(m_1)^{x_1}$ and $\sigma_2 = H(m_2)^{x_2}$ be two valid and regular BGLS signatures for two claims $c_1 = (\mathsf{pk}_1, m_1)$ and $c_2 = (\mathsf{pk}_2, m_2)$ with $\mathsf{pk}_1 = g^{x_1}$ and $\mathsf{pk}_2 = g^{x_2}$. Set

$$\sigma_1' \coloneqq g \cdot \sigma_1 \text{ and } \sigma_2' \coloneqq g^{-1} \cdot \sigma_2.$$

The signatures σ_1' and σ_2' are clearly not valid and therefore also not regular for the claim sequences (c_1) and (c_2). But if we aggregate them, it holds that

$$\mathsf{Agg}((c_1), (c_2), \sigma_1', \sigma_2') = g \cdot g^{-1} \cdot \sigma_1 \cdot \sigma_2 = \sigma_1 \cdot \sigma_2,$$

which is a regular and valid aggregate signature for the claim sequence (c_1, c_2), even though the individual signatures were irregular.

However, this does not contradict security, as crafting such signatures that "cancel out" their irregularity is hard if one does not know fitting regular signatures σ_i, otherwise the scheme would not be unforgeable. Suppose an attacker $\mathcal{A}$ never queries a signature for a message m^* under his challenge public key pk^*. Suppose further that $\mathcal{A}$ is able to create an aggregate signature τ, such that

$$R \subsetneq \mathsf{Vfy}(C, \tau) \text{ and } c^* \coloneqq (\mathsf{pk}^*, m^*) \in \mathsf{Vfy}(C, \tau) \setminus R,$$

i.e. τ was created by aggregating a signature that is irregular for c^*, but c^* nonetheless appears in the output of Vfy. Then τ would clearly be a forgery, since the AS-EUF-CMA security definition does not require that the attacker outputs a regular signature. The validity of c^* under τ and the "freshness" of m^* (i.e. $\mathcal{A}$ never asked for a signature on m^*) are the only requirements for τ to be counted as a winning forgery, which both would be fulfilled here. So, attackers cannot use this "canceling out" property to forge signatures.

Still, the attacker might be able to easily output an aggregate signature where $R \subsetneq \mathsf{Vfy}(C, \tau)$, since he can freely choose additional secret keys and could aggregate irregular signatures under these keys which "cancel out" their irregularity, like in Example 4.3.8 for the BGLS scheme. While this would not be a valid forgery (since the attacker chose the keys himself), this shows that unforgeability does not imply the direction $R \supseteq \mathsf{Vfy}(C, \tau)$.

While this means that Vfy might output "false positives", we can still be sure that these claims must have been validly signed at some point.[3] Observe furthermore that by definition of aggregate signature schemes with list verification, Vfy will *never* output any claim that was *not* part of its input. Even if we would not explicitly require this in Definition 4.3.1, any Vfy algorithm can easily be changed so that it never outputs any such claim. One could simply define a Vfy' algorithm that only outputs the claims returned by Vfy that were also part of the input. A change like this to the scheme in question has no influence on its fault-tolerance, correctness or security.

[3] In fact, it could be interesting to investigate whether this property of "canceling out" irregularity could be used to "repair" partially invalid aggregate signatures of a fault-tolerant scheme. However, we leave this question open for now.

Definition 4.3.9 (Fault-Tolerant Aggregate Signature Scheme)**.** *A* d-fault-tolerant aggregate signature scheme *is an aggregate signature scheme with list verification that is tolerant against* d *errors. A* fault-tolerant aggregate signature scheme *is a scheme that is* d*-fault-tolerant for some* $d > 0$*.*

Note that fault-tolerance is an unconditional property and that we do not need to resort to complexity theoretic hardness assumptions to prove it. Note furthermore that 0-fault-tolerance means that if $M = \{(c_1, \tau_1), \ldots, (c_n, \tau_n)\}$ contains only regularly created signatures, then Vfy must output all claims in M (or the corresponding claim sequence C, respectively). This is analogous to the common definition of correctness for aggregate signature schemes, which motivates the following definition of correctness:

Definition 4.3.10 (Correctness for Aggregate Signatures with List Verification)**.** *An aggregate signature scheme with list verification is* correct*, if it is tolerant against* 0 *errors.*

Remark. Our definitions above assume that aggregation is always done correctly. This is a necessary assumption, since it is impossible to give guarantees for all arbitrary errors that might happen during aggregation. Consider for example a faulty implementation of the Agg algorithm that sometimes outputs a random string or a malicious signer that ignores the aggregate-so-far and outputs a random signature. It is an interesting open question to find a fault-tolerant signature scheme that can tolerate certain types of aggregation errors, too.

4.4 Fault-Tolerance and Signature Size

A typical and desired requirement for aggregate signature schemes is that the size of an aggregate signature is constant and as close as possible to the size of individual signatures, no matter how many signatures are aggregated. Ideally, aggregate and individual signatures should have the same size. Also, the number of signatures that can be aggregated into a single signature should be unbounded.

These requirements are fulfilled by almost all known sequential, synchronized and fully flexible schemes, but unfortunately, it is *not possible* for fault-tolerant schemes to fulfill them. They *cannot* offer a signature size which is independent of the number of individual signatures. In other words, we cannot hope to aggregate an unbounded number of individual signatures in a way that tolerates a constant number of errors and still receive a constant-size aggregate.

This follows from an information-theoretic argument: Assume the size of an aggregate signature is fixed to a constant number of l bits. This l-bit string then needs to be used by the verification algorithm as the only "source of information" to determine which of its input claims are valid. Hence, based on the l-bit string, the algorithm can distinguish at most 2^l different outputs. However, considering n claims and corresponding individual signatures, d of which are irregular, there are $\binom{n}{n-d}$ possible different subsets (and thus outputs) which should be distinguishable by the verification algorithm by considering this string. So n is upper bounded by $\binom{n}{n-d} \leq 2^l$. Therefore, if the number n of claims that can be aggregated should be unbounded, then the size of aggregate signatures necessarily needs to grow. Next, we show this formally for "ideal" fault-tolerant aggregate signature schemes, where Vfy always outputs exactly all claims that were signed by regular signatures.

Theorem 4.4.1. *Let $n, d \in \mathbb{N}$ and $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ be a d-fault-tolerant signature scheme. Assume that $\mathsf{Vfy}(C, \tau) = R$ for all claim sequences C and corresponding signatures τ constructed from an arbitrary multiset $M = \{(c_1, \tau_1), \ldots, (c_n, \tau_n)\}$ of n claim–signature pairs that contains at most $d \leq n$ errors and where R is the multiset of all claims c_i accompanied by a regular signature τ_i in M.*

Then it holds that $|\tau| \in \Omega(\log_2 n)$ as a function of n, where d is considered constant and $|\tau|$ is the size of the signature τ in bits.

Proof. Call an output O of Vfy *in accordance with* C, if O is a sub-multiset of $\mathsf{elem}(C)$ and $|O| \geq |\mathsf{elem}(C)| - d$.

Clearly, since we assumed that Vfy always outputs R, the output of Vfy must be in accordance with C. For a fixed number of errors $i \in \{0, \ldots, d\}$, there are $\binom{n}{i}$ distinct outputs in accordance with C. Thus, for up to d errors, there are up to

$$s(n) \coloneqq \sum_{i=0}^{d} \binom{n}{i} \geq \binom{n}{d} \geq \frac{(n-d)^d}{d!}$$

distinct outputs in accordance with C. Vfy must use τ to determine the *correct* output R among the set of outputs in accordance with C. If the signature size $|\tau|$ is at most $l \in \mathbb{N}$ bits, then Vfy can distinguish at most 2^l cases based on τ. Thus, we must have $2^l \geq s(n)$, or, equivalently,

$$l \geq \log_2 s(n) \geq \log_2 \frac{(n-d)^d}{d!} = d \log_2(n-d) - \log_2(d!) \in \Omega(\log_2 n).$$

□

Note again that the assumption that $\mathsf{Vfy}(C, \tau) = R$ is somewhat artificial. On the one hand, $R \subseteq \mathsf{Vfy}(C, \tau)$ is required by the definition of fault-tolerance. Intuitively, one would expect the other direction $R \supseteq \mathsf{Vfy}(C, \tau)$ to follow from the unforgeability of Σ. However, this is not the case.

As already discussed, the aggregation of multiple irregular signatures can "cancel out" their irregularity and it can be easy for an attacker to compute such signatures, since he can choose almost all secret keys himself (see the paragraphs below the Definitions 4.3.5 and 4.3.7). Furthermore, security is only required against attackers that have polynomial run time in the security parameter κ, i.e. attackers that can create at most a polynomial number of claims.

Still, the arguments presented also apply to "non-ideal" schemes where d is set to the number of claims accompanied by irregular signatures which irregularity does *not get canceled out* by aggregating them with the other signatures.

Furthermore, it is unclear how "non-ideality" could help to improve this lower bound, since the output of the verification algorithm must always be meaningful, i.e. it must at least output all claims accompanied by a regular signature, but cannot simply output all claims or some arbitrary set of claims if the scheme is supposed to be unforgeable.

Since the size of an aggregate signature cannot be constant for fault-tolerant schemes, it is important to analyze the *compression ratio* of such schemes, which captures how an aggregate signature may grow in size. The higher the compression ratio, the smaller the resulting aggregate. Therefore the compression ratio is a measure of quality of a fault-tolerant aggregate signature scheme.

Definition 4.4.2 (Compression Ratio)**.** *Denote by* $\mathsf{size}(\sigma)$ *the size of a signature* σ*. Let* C *be a claim sequence of length* n *and* σ^* *an aggregate signature of maximum size*[4] *which is regular for* C*. We say that an aggregate signature scheme has* compression ratio $\rho(n)$ *if for all* C *and* σ^* *it holds that*

$$\frac{n}{\mathsf{size}(\sigma^*)} \in \Theta(\rho(n)).$$

Example 4.4.3. If the size of all aggregate signatures is upper bounded by a constant (e.g. if all aggregates have the same size as individual signatures), then the compression ratio is $\rho(n) = n$, which is optimal for common aggregate signature schemes, but not possible for fault-tolerant aggregate signatures, as argued above. If the size of all aggregate signatures grows linear in the number of aggregated signatures (e.g. if the Agg algorithm simply stores the individual signatures in a list), then no compression occurs and $\rho(n) = 1$.

4.5 A Construction from Cover-Free Families

We now present our generic construction of fault-tolerant aggregate signature schemes. It is based on an arbitrary aggregate signature scheme, which is used in a black-box way, and a cover-free family. Our scheme inherits its security from the used aggregate signature scheme, has a tight security proof and can tolerate d faults if a d-cover-free family is used.

Before we present the construction, we want to elaborate on the restrictions of the scheme that were already briefly mentioned in Section 4.1.1:

Bounded Aggregation. Our basic scheme can only be used to aggregate a bounded number of signatures. The bound is dependent on the cover-free family: The maximum number of signatures that can be aggregated is equal to the number of columns of the incidence matrix of the used cover-free family. In Section 4.5.1 we discuss how to remove this restriction.

Flexibility of Aggregation. Our scheme does not support fully flexible aggregation, but rather a slightly restricted form, which requires an order among the claims in the claim sequence, i.e. each claim is assigned to a fixed position in the sequence. The order of aggregation itself can be completely arbitrary, but the positions of the claims must be maintained by the aggregation and verification algorithms. Signatures can be aggregated in any order, as long as the claims are assigned to the correct positions in the claim sequence.

More precisely, when an individual signature τ' for a claim c is first aggregated into an aggregate signature τ, one must assign a unique "position" j to c. If one wishes to verify τ, one must call Vfy with a sequence of claims C that has c at its j-th position, i.e. $C[j] = c$. Two aggregate signatures τ_1, τ_2 for two claim sequences C_1, C_2 *cannot* be aggregated if for some index j both $C_1[j]$ and $C_2[j]$ are not empty, even if $C_1[j] = C_2[j]$.

However, if the signers agree in advance on distinct positions j of their claims, they can always aggregate their signatures into a single combined signature τ.

[4] The size of aggregate signatures might vary. For example, it might depend on the order of aggregation.

Furthermore, the signers do not need to uphold a fixed order of aggregation, the only requirement is that each signer uses his assigned position in the claim sequence and no other position. The resulting (aggregate) signatures themselves can then be aggregated in any order.

This requirement can easily be fulfilled in many applications, especially if the application itself requires some form of order or architecture between the signing parties, like in wireless sensor networks. Here, each sensor simply needs to be configured to use a different position j. If such an order is possible, than aggregation is (almost) as flexible as in a fully flexible scheme.

Moreover, it is always possible to use our scheme as a sequential aggregate signature scheme, since the position j of a claim needs only be determined when it is first aggregated. Our scheme is therefore suitable for all applications where sequential aggregate signatures are sufficient, too, such as secure logging [MT09]. Still, we would like to stress that our construction offers a much more flexible type of aggregation than just sequential aggregation.

Our form of aggregation cannot be directly compared with synchronized aggregation. Synchronized aggregation allows for fully flexible aggregation of signatures created using the same synchronizing information, but other signatures cannot be aggregated at all. In contrast, in our scheme only signatures that already have claims at the same position of their claim sequences cannot be aggregated. Synchronized schemes are however suitable for our construction, since if the parties are able to agree on some form of synchronization, they should also be able to agree on the positions of their claims.

For the above mentioned reasons, we have to deal with "incomplete" claim sequences, i.e. claim sequences that do not yet contain claims at all positions. When aggregating the signatures of two such incomplete claim sequences C_1, C_2, the claim sequences will be *merged*, meaning that claim placeholders in C_1 are replaced by actual claims from C_2, for each position j where $C_1[j] = \perp$ and $C_2[j] \neq \perp$, and vice versa (see Definition 3.1.3).

For technical reasons, we also require that there is no position where C_1 and C_2 both contain a claim, even if the claims are identical. As a consequence, if an individual signature τ is aggregated into two different aggregate signatures τ_1, τ_2 using the same position j, τ_1 and τ_2 cannot be aggregated later. Note, however, that τ could be aggregated with τ_1 and τ_2 at different positions, which would then also enable the aggregation of τ_1 and τ_2 at a later point.

Our Black-Box Construction. For the rest of this chapter, let $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Agg}', \mathsf{Vfy}')$ be an ordinary aggregate signature scheme and let $\mathcal{M}$ be the incidence matrix of a d-cover-free family $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$, as defined in Section 2.3. Let $\Sigma_{\mathsf{FT}} = (\mathsf{Gen}_{\mathsf{FT}}, \mathsf{Sign}_{\mathsf{FT}}, \mathsf{Agg}_{\mathsf{FT}}, \mathsf{Vfy}_{\mathsf{FT}})$ be the fault-tolerant aggregate signature scheme that we construct from Σ' and $\mathcal{F}$.

In our scheme Σ_{FT}, signatures for just one claim are simply signatures of the underlying scheme Σ', whereas aggregate signatures are short vectors of signatures of Σ'. Our construction uses the incidence matrix $\mathcal{M}$ of $\mathcal{F}$ to build these vectors. We identify each element of the universe $\mathcal{S}$ with a position in this vector and each subset $B \in \mathsf{Blocks}$ with an individual signature of the underlying scheme Σ'. To assign a position i to a claim c, we transform it to a claim sequence $C = (\perp, \ldots, \perp, c, \perp, \ldots, \perp)$ of length $\mathrm{cols}(\mathcal{M})$, such that $C[i] = c$. We therefore require that the underlying scheme Σ' supports claim placeholders as

an input to Agg' and Vfy'. However, this is not an essential restriction, as any normal aggregate scheme may be easily adapted to a scheme of the modified syntax, for example by ignoring any order and claim placeholders by applying $\mathsf{elem}(C)$ on the input claim sequences C before it is passed on to the Agg' and Vfy' algorithms.

We assume that $\mathsf{Agg}_{\mathsf{FT}}$ and $\mathsf{Vfy}_{\mathsf{FT}}$ have access to the incidence matrix $\mathcal{M}$. This can be realized by storing $\mathcal{M}$ as a "public parameter", so that all parties can access it. Alternatively, the algorithms could also construct the matrix (or only the needed parts of it) "on they fly", dependent on which option is better suited to the intended application of Σ_{FT}, which presents a trade-off between memory usage and performance.[5] Note that $\mathcal{M}$ could also be stored in the public keys, but this is the most wasteful option, since it would greatly and unnecessarily increase the size of all public keys. We further assume that all claim sequences have length $\mathrm{cols}(\mathcal{M})$. Next, we describe the algorithms of our fault-tolerant aggregate signature $\Sigma_{\mathsf{FT}} = (\mathsf{Gen}_{\mathsf{FT}}, \mathsf{Sign}_{\mathsf{FT}}, \mathsf{Agg}_{\mathsf{FT}}, \mathsf{Vfy}_{\mathsf{FT}})$.

- $\mathsf{Gen}_{\mathsf{FT}}(1^\kappa)$ creates and outputs a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}'(1^\kappa)$.
- $\mathsf{Sign}_{\mathsf{FT}}(\mathsf{sk}, m)$ computes $\sigma \leftarrow \mathsf{Sign}'(\mathsf{sk}, m)$ and returns σ.
- $\mathsf{Agg}_{\mathsf{FT}}(C_1, C_2, \tau_1, \tau_2)$ takes as input two claim sequences C_1 and C_2 and corresponding signatures τ_1 and τ_2. Then, it proceeds as follows:
  1. If C_1 and C_2 are *not* exclusively mergeable it aborts.
  2. If one or both of the claim sequences C_k ($k \in \{1, 2\}$) contains only one non-empty claim $c = C_k[j] \neq \perp$ ($j \in [\mathrm{rows}(\mathcal{M})]$), i.e. τ_k is an individual signature, then it sets $\sigma_k \coloneqq \tau_k$ and expands τ_k to a vector of length $\mathrm{rows}(\mathcal{M})$, by setting
  $$\tau_k[i] \coloneqq \begin{cases} \sigma_k, & \text{if } \mathcal{M}[i,j] = 1, \\ \lambda, & \text{otherwise,} \end{cases} \qquad \text{for } i = 1, \ldots, \mathrm{rows}(\mathcal{M}),$$
  where j is the index of c in the claim sequence C_k.
  3. Then the signatures τ_1, τ_2, which are both vectors now, are aggregated component-wise using Agg', i.e. it initializes a new vector τ and sets
  $$\tau[i] \leftarrow \mathsf{Agg}'(C_1[\mathcal{M}_i], C_2[\mathcal{M}_i], \tau_1[i], \tau_2[i]).$$
  for $i \in [\mathrm{rows}(\mathcal{M})]$.[6]

  Finally, $\mathsf{Agg}_{\mathsf{FT}}$ outputs τ.
- $\mathsf{Vfy}_{\mathsf{FT}}(C, \tau)$ takes as input a claim sequence C and an aggregate signature τ for C. For each component $\tau[i]$ of τ it computes $b_i \coloneqq \mathsf{Vfy}'(C[\mathcal{M}_i], \tau[i])$ and outputs the multiset of valid claims
$$C_{\text{valid}} \coloneqq \mathsf{elem}\left(\bigsqcup_{i \in [\mathrm{rows}(\mathcal{M})], b_i = 1} C[\mathcal{M}_i] \right).$$

[5] In the instantiation of our construction using polynomials over a finite field, it is easy to compute the whole matrix $\mathcal{M}$ or just parts of it (see Section 4.5.3).

[6] Note that here it can happen that one or both claim sequences $C_k[\mathcal{M}_i]$ only contain $\perp$ and that $\tau_k[i] = \lambda$. In this case, we assume that Agg' returns the signature of the claim sequence that contains at least one non-empty claim $c \neq \perp$ or the empty signature λ if both sequences contain only $\perp$ and both signatures are equal to λ.

Example 4.5.1. Let $\mathcal{M}$ be the incidence matrix used to instantiate our construction:

$$\mathcal{M} := \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Let c_1, c_2, c_2', c_3, c_4 be five distinct claims with:

- $\tau_1, \tau_2, \tau_2', \tau_3$ are individual regular signatures for c_1, c_2, c_2', c_3, respectively.
- τ_4 is an individual *invalid* (and therefore irregular) signature for c_4.
- Each claim gets associated with the position in the claim sequence equal to its index, i.e. c_1 with position 1, c_2' with position 2 etc.
- Let C_i be the corresponding claim sequences of length $\text{cols}(\mathcal{M}) = 6$ with the claim c_i at its correct position, i.e. $C_1 = (c_1, \perp, \perp, \perp, \perp, \perp)$, $C_2' = (\perp, c_2', \perp, \perp, \perp)$ and so on.

For ease of exposition, let "$\tau_i \cdot \tau_j$" denote $\mathsf{Agg}'(C_i, C_j, \tau_i, \tau_j)$. Now, τ_1 and τ_2 can be aggregated, since C_1 and C_2 are exclusively mergeable. $\mathsf{Agg}_{\mathsf{FT}}(C_1, C_2, \tau_1, \tau_2)$ will output an aggregate signature $\tau_{1,2}$ for the claim sequence $C_{1,2} = C_1 \sqcup C_2$ $= (c_1, c_2, \perp, \perp, \perp, \perp)$ with

$$\tau_{1,2} = \begin{pmatrix} \tau_1 \\ \tau_1 \cdot \tau_2 \\ \tau_2 \\ \lambda \end{pmatrix}.$$

$\mathsf{Vfy}_{\mathsf{FT}}(C_{1,2}, \tau_{1,2})$ would do the following for each component of the vector $\tau_{1,2}$:

Component 1: Check that $\mathsf{Vfy}'((c_1, \perp, \perp, \perp, \perp, \perp), \tau_1) = 1$.

Component 2: Check that $\mathsf{Vfy}'((c_1, c_2, \perp, \perp, \perp, \perp), \tau_1 \cdot \tau_2) = 1$.

Component 3: Check that $\mathsf{Vfy}'((\perp, c_2, \perp, \perp, \perp, \perp), \tau_2) = 1$.

Component 4: Check that $\mathsf{Vfy}'((\perp, \perp, \perp, \perp, \perp, \perp), \lambda) = 1$.

$\mathsf{Vfy}_{\mathsf{FT}}$ would then return c_1 and c_2, since all checks would succeed. The aggregate signature $\tau_{1,2}$ could further be aggregated with τ_3, resulting in a signature $\tau_{1,2,3}$ valid for $C_{1,2,3} = (c_1, c_2, c_3, \perp, \perp, \perp)$ with

$$\tau_{1,2,3} = \begin{pmatrix} \tau_1 \\ \tau_1 \cdot \tau_2 \\ \tau_2 \cdot \tau_3 \\ \tau_3 \end{pmatrix}.$$

Since $C_{1,2,3}[2] \neq \perp \neq C_2'[2]$, the two claim sequences $C_{1,2,3}$ and C_2' are *not* exclusively mergeable and *cannot* be aggregated. However, c_2' could be associated with another, still empty, position like position 4 and then be aggregated to $C_{1,2,3}$.

$C_{1,2,3}$ and C_4 could be aggregated, resulting in $\tau_{1,2,3,4}$ for $C_{1,2,3,4} = (c_1, c_2, c_3, c_4, \perp, \perp)$ with one error:

$$\tau_{1,2,3,4} = \begin{pmatrix} \tau_1 \cdot \tau_4 \\ \tau_1 \cdot \tau_2 \\ \tau_2 \cdot \tau_3 \cdot \tau_4 \\ \tau_3 \end{pmatrix}.$$

$\mathsf{Vfy}_{\mathsf{FT}}(C_{1,2,3,4}, \tau_{1,2,3,4})$ would do the following:

Component 1: Check that $\mathsf{Vfy}'((c_1, \perp, \perp, c_4, \perp, \perp), \tau_1 \cdot \tau_4) = 1$, which would *fail*, since τ_4 is invalid.

Component 2: Check that $\mathsf{Vfy}'((c_1, c_2, \perp, \perp, \perp, \perp), \tau_1 \cdot \tau_2) = 1$.

Component 3: Check that $\mathsf{Vfy}'((\perp, c_2, c_3, c_4, \perp, \perp), \tau_2 \cdot \tau_4) = 1$, which would *fail*, since τ_4 is invalid.

Component 4: Check that $\mathsf{Vfy}'((\perp, \perp, c_3, \perp, \perp, \perp), \tau_3 = 1$.

Now, all checks containing c_4 fail and $\mathsf{Vfy}_{\mathsf{FT}}$ would therefore not output it. However, the second and fourth checks still succeed and Vfy would therefore successfully output c_1, c_2 and c_3, even though one faulty signature was aggregated.

The next two theorems show that if Σ' is unforgeable, then so is Σ_{FT} and that Σ_{FT} is d-fault-tolerant if $\mathcal{F}$ is a d-cover-free family.

Theorem 4.5.2. *If Σ' is a (t', q, ε)-AS-EUF-CMA secure aggregate signature scheme, then the scheme Σ_{FT} defined above is a (t, q, ε)-AS-EUF-CMA secure aggregate signature scheme with list verification, where t is approximately the same as t'.*

Proof. Assume that $\mathcal{A}$ is an attacker breaking the (t, q, ε)-AS-EUF-CMA security of Σ_{FT}. We construct a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to break the (t', q, ε)-AS-EUF-CMA security of Σ'.

In the Setup Phase, $\mathcal{B}$ receives a public key pk from its challenger, which it passes on to $\mathcal{A}$. Whenever $\mathcal{A}$ makes a signature query for a message m, $\mathcal{B}$ obtains a signature σ by forwarding m to the challenger. $\mathcal{B}$ then sends σ to $\mathcal{A}$ and continues the simulation. Since individual signatures of Σ_{FT} are simply signatures of Σ', this is a valid answer to the query. At some point, $\mathcal{A}$ outputs a claim sequence C^* and a signature τ^* as its forgery. $\mathcal{B}$ now checks that C^* contains a claim $c^* = (\mathsf{pk}, m^*)$ for a message m^* that $\mathcal{A}$ never asked a signature for in the Query Phase and that c^* is in the output of $\mathsf{Vfy}_{\mathsf{FT}}(C^*, m^*)$.

If this is not the case, then the attack of $\mathcal{A}$ was not successful and $\mathcal{B}$ terminates by outputting $\perp$. Otherwise, by definition of $\mathsf{Vfy}_{\mathsf{FT}}$, there must be an index i such that

$$c^* \in \mathsf{elem}(C^*[\mathcal{M}_i]) \text{ and } \mathsf{Vfy}'(C^*[\mathcal{M}_i], \tau^*[i]) = 1.$$

Note that the queries of $\mathcal{B}$ are exactly the same as those of $\mathcal{A}$, so if $\mathcal{A}$ did not query m^*, then neither did $\mathcal{B}$. Therefore, $\mathcal{B}$ can output $C^*[\mathcal{M}_i]$ and $\tau^*[i]$ to win the experiment.

Thus, $\mathcal{B}$ wins exactly if and only if $\mathcal{A}$ wins. Therefore $\mathcal{B}$ also has success probability ε. We also see that $\mathcal{B}$ makes at most q queries. Furthermore, the run time of $\mathcal{B}$ is approximately the same as the run time of $\mathcal{A}$. □

Theorem 4.5.3. *If $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$ is a d-cover-free family, then the aggregate signature scheme Σ_{FT} defined above is tolerant against d errors, and in particular, it is correct.*

Proof. Let $M = \{(c_1, \tau_1), \ldots, (c_n, \tau_n)\}$ be a multiset of claim-signature pairs, which is partitioned into two multisets M_{irreg} and M_{reg}, containing the pairs (c_i, τ_i) for which τ_i is irregular or regular for c_i, respectively. Let M contain at most d errors, i.e. $|M_{\text{irreg}}| \leq d$. Moreover, let τ be a signature that was aggregated from the signatures in M in arbitrary order and C the corresponding claim sequence.

To simplify the proof and without loss of generality, we assume that $C = (c_1, \ldots, c_n)$ contains no claim placeholders and that the order in the claim sequence is the same as in the indexing of the signatures in M. Finally, let $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$ be the d-cover-free family used by Σ_{FT}, where $\mathcal{S} = \{s_1, \ldots, s_m\}$ and $\mathsf{Blocks} = \{B_1, \ldots, B_n\}$.

We need to show that $R \subseteq \mathsf{Vfy}_{\mathsf{FT}}(C, \tau) =: V$, where R is the multiset of all the claims in M_{reg}. Recall that $\text{rows}(\mathcal{M})$ and $\text{cols}(\mathcal{M})$ denote the number of rows and columns of $\mathcal{M}$, respectively. Let $b_i := \mathsf{Vfy}'(C[\mathcal{M}_i], \tau[i])$ for all $i \in [\text{rows}(\mathcal{M})]$.

Assume for a contradiction that there is a claim c^* that is contained strictly more often in R than in V. Then there must exist an index j^* such that $C[j^*] = c^*$ and $b_i = 0$ for all indices $i \in [\text{rows}(\mathcal{M})]$ with $\mathcal{M}[i, j^*] = 1$. In the following, let

$$I := \{i \in [\text{rows}(\mathcal{M})] \colon \mathcal{M}[i, j^*] = 1\}$$

be the set of these indices. Observe that these are the indices of all rows where the individual signature τ^* for c^* is aggregated into $\tau[i]$.

We now show that this implies that the set B_{j^*}, which corresponds to the column j^* of $\mathcal{M}$, is covered by the sets B_k, corresponding to the columns of the claims with irregular signatures, which is a contradiction to the d-cover-freeness of $\mathcal{F}$, since $|M_{\text{irreg}}| \leq d$.

For each $i \in I$, since $b_i = 0$ and using the correctness of Σ', there must be some $k \in [n]$ such that $(c_k, \sigma_k) \in M_{\text{irreg}}$ and $\mathcal{M}[i, k] = 1$. Let K denote the set of these indices. Since M contains at most d errors, there are at most d such indices k in total, i.e. we have $|K| \leq d$. Note furthermore that $j^* \notin K$, since $(c^*, \tau^*) \in M_{\text{reg}}$, according to our assumption.

We now have established that for each $i \in I$, there exists a k with $(c_k, \sigma_k) \in M_{\text{irreg}}$ and $\mathcal{M}[i, k] = 1$. We have also shown that $|K| \leq d$ and $j^* \notin K$. Recall that by definition of the incidence matrix $\mathcal{M}$, we have for all $i \in [\text{rows}(\mathcal{M})]$ and $j \in [\text{cols}(\mathcal{M})]$:

$$\mathcal{M}[i, j] = 1 \quad \iff \quad s_i \in B_j.$$

Restating the facts from the above paragraph using this equivalence yields that for all i with $s_i \in B_{j^*}$, there exists a $k \neq j^*$ with $s_i \in B_k$, where there are at most d distinct indices $k \in K$ in total. But this means that $B_{j^*} \subseteq \bigcup_{k \in K} B_k$, where the union is over at most d different subsets $B_k \neq B_{j^*}$ of $\mathcal{S}$. This is a direct contradiction to the d-cover-freeness of $\mathcal{F}$.

So our assumption must be false and there cannot exist a claim c^* which is contained strictly more often in R than in V. We therefore have $R \subseteq V$, which concludes the proof. □

Compression Ratio. Let C be a claim sequence of length $n \in \mathbb{N}$ and τ be an aggregate signature of Σ_{FT} regular for C. We assume in the following that the size of all signatures of the underlying scheme Σ' is bounded by a constant s and is at least 1. Then the compression ratio of our scheme is $\rho(n) = \frac{n}{\text{rows}(\mathcal{M})}$, since

$$\begin{aligned} \frac{n}{\mathsf{size}(\tau)} &\leq \frac{n}{\text{rows}(\mathcal{M}) \cdot s} \in \mathcal{O}(\rho(n)) \text{ and} \\ \frac{n}{\mathsf{size}(\tau)} &\geq \frac{n}{\text{rows}(\mathcal{M})} \quad \in \Omega(\rho(n)). \end{aligned} \tag{4.1}$$

Clearly, the compression ratio $\rho(n)$ of Σ_{FT} is less than 1 if $n < \text{rows}(\mathcal{M})$, so aggregate signatures are actually larger than the sum of the sizes of the individual signature when only a few signatures have been aggregated so far (i.e. less then $\text{rows}(\mathcal{M})$ individual signatures). The scheme can be easily adapted to fix this behavior by simply storing the first $\text{rows}(\mathcal{M})$ individual signatures $\tau_1, \ldots, \tau_{\text{rows}(\mathcal{M})}$ in a list, instead of immediately aggregating them. As soon as the signature $\tau_{\text{rows}(\mathcal{M})+1}$ is added, the individual signatures are aggregated using the aggregation algorithm defined above. When further signatures are added, the size of the aggregate signature remains bounded by $\text{rows}(\mathcal{M}) \cdot s$.

4.5.1 Achieving Unbounded Aggregation

In order to achieve unbounded aggregation, we do not need just one cover-free family, but a sequence of cover-free families increasing in size, such that we can jump to the next larger one as soon as we exceed the capacity for the bound of signatures that can be aggregated. This sequence needs to exhibit a monotonicity property, in order to work with our scheme:

Definition 4.5.4 (Monotone Cover-Free Family)**.** *Let $(\mathcal{M}^{(l)})_l$ be a family of incidence matrices of corresponding d-cover-free families $(\mathcal{F}_l)_l := (\mathcal{S}_l, \mathsf{Blocks}_l)_l$, where* $\text{rows}(l)$ *denotes the number of rows and* $\text{cols}(l)$ *denotes the number of columns of $\mathcal{M}^{(l)}$. $(\mathcal{M}^{(l)})_l$ is a* monotone family of incidence matrices *of $(\mathcal{F}_l)_l$, if for $l \geq 1$ we have $\mathcal{S}_l \subseteq \mathcal{S}_{l+1}$, $\mathsf{Blocks}_l \subseteq \mathsf{Blocks}_{l+1}$, such that*

$$\begin{aligned} \mathcal{S}_{l+1} &= \{s_1, \ldots, s_{\text{rows}(l)}, s_{\text{rows}(l)+1}, \ldots, s_{\text{rows}(l+1)}\} \text{ and} \\ \mathsf{Blocks}_{l+1} &= \{B_1, \ldots, B_{\text{cols}(l)}, B_{\text{cols}(l)+1}, \ldots, B_{\text{cols}(l+1)}\}, \end{aligned}$$

where $\mathcal{S}_l = \{s_1, \ldots, s_{\text{rows}(l)}\}$ and $\mathsf{Blocks}_l = \{B_1, \ldots, B_{\text{cols}(l)}\}$. If this is the case, we also call $(\mathcal{F}_l)_l$ a monotone cover-free family*.*

Note that Definition 4.5.4 implies for monotone cover-free families, that

$$\mathcal{M}^{(l+1)} = \begin{pmatrix} \mathcal{M}^{(l)} & \mathbf{A} \\ \mathbf{0} & \mathbf{B} \end{pmatrix}$$

where for $i = 1, \ldots, \text{rows}(l), j = \text{cols}(l) + 1, \ldots, \text{cols}(l+1)$

$$\mathbf{A}[i,j] = \begin{cases} 1, & \text{if } s_i \in B_j, \\ 0, & \text{otherwise} \end{cases}$$

and for $i = \text{rows}(l), \ldots, \text{rows}(l+1), j = \text{cols}(l) + 1, \ldots, \text{cols}(l+1)$

$$\mathbf{B}[i,j] = \begin{cases} 1, & \text{if } s_i \in B_j, \\ 0, & \text{otherwise.} \end{cases}$$

So, each $\mathcal{M}^{(l)}$ contains all previous $\mathcal{M}^{(1)}, \ldots, \mathcal{M}^{(l-1)}$. Now, we can achieve unbounded aggregation by replacing the fixed incidence matrix $\mathcal{M}$ of a d-cover-free family in our construction with a monotone cover-free family with incidence matrices $(\mathcal{M}^{(l)})_l$. For this, the aggregation algorithm $\mathsf{Agg}_{\mathsf{FT}}$ on inputs C_1, C_2, τ_1, τ_2 first has to determine the smallest l, such that $\text{cols}(l) \geq \max(|C_1|, |C_2|)$ and then proceeds with the corresponding incidence matrix $\mathcal{M}^{(l)}$. Analogously, the verification algorithm $\mathsf{Vfy}_{\mathsf{FT}}$ on inputs C, τ first determines the smallest l such that $\text{cols}(l) \geq |C|$.

Compression Ratio. The compression ratio of our unbounded scheme is $\rho(n) = n/\text{rows}(l)$, where l is the minimum index such that $\text{cols}(l) \geq n$.

Remark. Idalino and Moura [IM18] improved on our unbounded construction by using *nested* cover-free families that offer better compression ratios (dependent on the number d of faults that can be tolerated).

4.5.2 Additional Feature: Selective Verification

Let τ be a signature which is regular for the claim sequence $C = (c_1, \ldots, c_n)$ of our fault-tolerant aggregate signature scheme. Assume we would want to know whether a signature for a specific claim c^* is valid under τ, but we want to avoid verifying all the claims in C to save verification time, especially if C is large. To solve this problem, an additional algorithm $\mathsf{SelectiveVfy}_{\mathsf{FT}}(C, \tau, c^*)$ can be added to our construction that outputs the number of occurrences[7] of c^* in $\mathsf{Vfy}_{\mathsf{FT}}(C, \tau)$ and which is faster to compute than actually computing $\mathsf{Vfy}_{\mathsf{FT}}(C, \tau)$ itself.

Let Σ_{FT} be the scheme defined by our construction to achieve fault-tolerance, Σ' the underlying aggregate signature scheme and $\mathcal{F}$ the cover-free family with incidence matrix $\mathcal{M}$ used to instantiate Σ_{FT}. Then $\mathsf{SelectiveVfy}_{\mathsf{FT}}$ works as follows:

1. Determine the set J of indices j where c^* occurs in C, i.e. $c_j = c^*$.
2. Determine the set $I := \{i \in \text{rows}(\mathcal{M}) \colon \mathcal{M}[i, j] = 1 \text{ for a } j \in J\}$, containing the indices of all rows where an individual signature for c^* should have been aggregated. Initialize $M := ()$.
3. Iterate over all $i \in I$. If $b_i := \mathsf{Vfy}'(C[\mathcal{M}_i], \tau[i]) = 1$ for an index i, set $M := M \sqcup C[\mathcal{M}_i]$. As soon as M contains $|J|$ occurrences of c^*, skip all remaining $i \in I$.[8]
4. After the loop is done, output the number of occurrences of c^* in M.

Since $\mathsf{Vfy}_{\mathsf{FT}}$ returns all claims that are contained in a subsequence $C[\mathcal{M}_i]$ with $b_i = 1$, the output of $\mathsf{SelectiveVfy}_{\mathsf{FT}}$ is exactly the number of occurrences of c^* in $\mathsf{Vfy}_{\mathsf{FT}}$. $\mathsf{SelectiveVfy}_{\mathsf{FT}}$ therefore inherits the fault-tolerance and security properties already proven for $\mathsf{Vfy}_{\mathsf{FT}}$.

[7] It outputs the number of occurrences, since the claim could be signed multiple times in the aggregate signature. If one is not interested in this number, then it is easy to reformulate the algorithm so that it stops once it finds a valid signature for the given claim and returns 1.

[8] The remaining i can safely be skipped, because the signature(s) of c^* might have been aggregated in more rows than we need to check. Suppose, for example, that c^* occurs only once in C. Then one valid signature $\tau[i]$ suffices to see that c^* was correctly signed.

In the best case, $\mathsf{SelectiveVfy}_{\mathsf{FT}}$ requires only one call to the underlying verification algorithm Vfy'. In the worst case, it still only requires $|I| \leq \sum_{j \in J} |B_j|$ calls to Vfy', where B_j is the set from the cover-free family corresponding to column j of $\mathcal{M}$.

Furthermore, it is even possible to create a "subsignature" for c^* that allows everyone to check that c^* has a valid signature without requiring the complete claim sequence C and the complete signature τ: It is sufficient to give $C' := \bigsqcup_{i \in I} C[\mathcal{M}_i]$ and the signatures $\tau[i]$ for $i \in I$ to the verifier.

4.5.3 Instantiation using a Cover-Free Family based on Polynomials over a Finite Field

There exist several constructions of d-cover-free families in the literature, for instance, constructions based on concatenated codes [LVY01; DMR00a; DMR00b], polynomials, algebraic-geometric Goppa codes, as well as randomized constructions [KRS99]. For concreteness, we show an instantiation of our construction using a cover-free family based on polynomials over a finite field.

Remark. This section is only a brief overview. This instantiation is discussed in more detail in the PhD thesis of Jessica Koch [Koc19] and in [Har+16].

We use a deterministic construction of a d-cover-free family based on polynomials like in [KRS99]. For that, let $q, k \in \mathbb{N}$. The numbers q and k are parameters of the cover-free family, but do not reflect a property of our construction.

For our d-cover-free family $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$ let $\mathbb{F}_q = \{x_1, \ldots, x_q\}$ be a finite field and

$$\mathcal{S} := \mathbb{F}_q^2 = \{(x_i, x_j) \colon i, j = 1, \ldots, q\}, \text{ with } |\mathcal{S}| = q^2.$$

For ease of presentation, we assume that q is a prime (as opposed to a prime power), so we may write $\mathbb{F}_q = \{0, \ldots, q-1\}$. We consider the set of all univariate polynomials $f \in \mathbb{F}_q[X]$ of degree at most k, denoted by $\mathbb{F}_q[X]_{\leq k}$, i.e.

$$\mathbb{F}_q[X]_{\leq k} := \left\{a_k X^k + \ldots + a_1 X + a_0 \colon a_i \in \mathbb{F}_q,\ i = 0, \ldots, k\right\}$$

with $|\mathbb{F}_q[X]_{\leq k}| = q^{k+1}$. Now, for every $f \in \mathbb{F}_q[X]_{\leq k}$, we consider the subsets

$$B_f = \{(x_1, f(x_1)), \ldots, (x_q, f(x_q))\} \subset \mathcal{S} \text{ of size } q,$$

consisting of all tuples $(x, y) \in \mathcal{S}$ which lie on the graph of $f \in \mathbb{F}_q[X]_{\leq k}$, i.e. for which $f(x) = y$. From this we obtain

$$\mathsf{Blocks} := \{B_f \colon f \in \mathbb{F}_q[X]_{\leq k}\},$$

which is of size q^{k+1}. For any distinct $B_f, B_{f_1}, \ldots, B_{f_d} \in \mathsf{Blocks}$ it holds that

$$|B_f \cap B_{f_i}| \leq k,$$

since the degree of each polynomial $g_i := f - f_i$ is at most k and hence each g_i has at most k zeros. Thus, we have

$$\left| B_f \setminus \bigcup_{i=1}^{d} B_{f_i} \right| \geq q - d \cdot k.$$

Therefore, to achieve a d-cover-free family with this construction, $q \geq d \cdot k + 1$ must be fulfilled.

Now, we consider the incidence matrix $\mathcal{M}$ of the d-cover-free family, which consists of $|\mathcal{S}|$ rows and $|\mathsf{Blocks}|$ columns. Each row corresponds to an element of $\mathcal{S}$ and each column to an element of Blocks. In the construction above each row corresponds to a tuple $(x, y) \in \mathbb{F}_q^2$, where the order is $(0,0), (0,1), \ldots, (q-1, q-1)$. In the following, let (x_i, y_i) denote the corresponding tuple for row i, $i = 0, \ldots, q^2 - 1$. We start counting from 0 for simplicity, hence,

$$\begin{aligned}
(x_0, y_0) &= (0,0), & \ldots, \quad & (x_{q-1}, y_{q-1}) = (0, q-1), \\
(x_q, y_q) &= (1,0), & \ldots, \quad & (x_{2q-1}, y_{2q-1}) = (1, q-1), \\
& & \ddots & \\
(x_{q^2-q}, y_{q^2-q}) &= (q-1, 0), & \ldots, \quad & (x_{q^2-1}, y_{q^2-1}) = (q-1, q-1).
\end{aligned}$$

Each column of the incidence matrix $\mathcal{M}$ corresponds to a polynomial of degree at most k, where we decide to start with constant polynomials and end with polynomials of degree k, i.e.

$$\begin{array}{cccc}
f_0 \coloneqq 0, & f_1 \coloneqq 1, & \ldots, & f_q \coloneqq X, \\
f_{q+1} \coloneqq X + 1, & f_{q+2} \coloneqq X + 2, & \ldots, & f_{2q} \coloneqq 2X, \\
f_{2q+1} \coloneqq 2X + 1, & f_{2q+2} \coloneqq 2X + 2, & \ldots, & \vdots \\
\vdots & \vdots & \ddots & \vdots \\
\ldots & \ldots & \ldots & f_{q^{k+1}-1} \coloneqq \sum_{i=0}^{k} (q-1) X^i.
\end{array}$$

By f_j we denote the corresponding polynomial for column j, for $j = 0, \ldots, q^{k+1} - 1$, again starting from 0. Now, the incidence matrix is built as

$$\mathcal{M}[i, j] = \begin{cases} 1, & \text{if } f_j(x_i) = y_i, \\ 0, & \text{otherwise.} \end{cases}$$

Remark. With this univariate polynomial-based construction of a d-cover-free family it is very easy to generate the incidence matrix or only some parts of it, so instead of storing $\mathcal{M}$, the algorithms of Σ_{FT} can construct it "on the fly". This is also helpful if we want to check some information separately (see Section 4.5.2). If, for example, one is interested in verifying the validity of only one single claim–signature pair (c_j, σ_j) in an aggregate signature, it is not necessary to generate the whole matrix but only the rows where the related column j has 1-entries. One simply needs to compute the polynomial that corresponds to column j. For more details on how this can be done efficiently, see [Har+16; Koc19].

Compression Ratio of Our Bounded Scheme. If our bounded scheme is instantiated with this cover-free family and we assume that the size of signatures of the underlying scheme Σ' is bounded by a constant s, then, as shown for Σ_{FT} (see equation (4.1) in Section 4.5), the compression ratio is

$$\rho(n) = \frac{n}{\text{rows}(\mathcal{M})} = \frac{n}{|\mathcal{S}|} = \frac{n}{q^2} \ .$$

For $n = |\mathsf{Blocks}|$, we therefore have

$$\rho(n) = \frac{|\mathsf{Blocks}|}{|\mathcal{S}|} = \frac{q^{k+1}}{q^2} \ .$$

Since $q \geq d \cdot k + 1$, we have that $|\mathsf{Blocks}|$ grows exponentially in k, whereas $|\mathcal{S}|$ grows only quadratically in k. Hence, $|\mathsf{Blocks}|$ is exponential in $|\mathcal{S}|$, or, stated differently, $|\mathcal{S}|$ is logarithmic in $|\mathsf{Blocks}|$. This means that the size of an aggregate signature is logarithmic in the number of aggregated individual signatures, once more than $\mathrm{rows}(\mathcal{M})$ signatures have been aggregated.

Compression Ratio of Our Unbounded Scheme. To instantiate our unbounded construction, we first need to construct a monotone cover-free family. The next lemma by [LVW06] shows how such a family can be generically constructed from any cover-free family:

Lemma 4.5.5. *If $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$ and $\mathcal{F}' = (\mathcal{S}', \mathsf{Blocks}')$ are d-cover-free families, then there exist a d-cover-free family $\mathcal{F}^* = (\mathcal{S}^*, \mathsf{Blocks}^*)$ with $|\mathcal{S}^*| = |\mathcal{S}| + |\mathcal{S}'|$ and $|\mathsf{Blocks}^*| = |\mathsf{Blocks}| + |\mathsf{Blocks}'|$.*

Proof. Suppose $\mathcal{M}$ and $\mathcal{M}'$ are the incidence matrices of d-cover-free-families $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$ and $\mathcal{F}' = (\mathcal{S}', \mathsf{Blocks}')$, respectively. Then

$$\mathcal{M}^* = \begin{pmatrix} \mathcal{M} & \mathbf{0} \\ \mathbf{0} & \mathcal{M}' \end{pmatrix}$$

is an incidence matrix for a d-cover-free family $\mathcal{F}^* = (\mathcal{S}^*, \mathsf{Blocks}^*)$ with $|\mathcal{S}^*| = |\mathcal{S}| + |\mathcal{S}'|$ and $|\mathsf{Blocks}^*| = |\mathsf{Blocks}| + |\mathsf{Blocks}'|$. □ □

If we now instantiate our unbounded scheme with the monotone family of cover-free families obtained by fixing an incidence matrix $\mathcal{M}$ and repeatedly using Lemma 4.5.5 on $\mathcal{M}$, then the asymptotic compression ratio is $\rho(n) = 1$, since

$$\frac{n}{\mathrm{rows}(l)} \leq \frac{\mathrm{cols}(l)}{\mathrm{rows}(l)} = \frac{l\,\mathrm{cols}(\mathcal{M})}{l\,\mathrm{rows}(\mathcal{M})} = \frac{\mathrm{cols}(\mathcal{M})}{\mathrm{rows}(\mathcal{M})} \quad \text{for all } l,$$

which is constant. Therefore, the size of an aggregate signature is linear in the length of the claim sequence.

However, if we assume that all signatures of the underlying scheme Σ' have a size bounded by s, then the concrete size of an aggregate signature is at most

$$\begin{aligned} \mathrm{rows}(l)s &\leq l\,\mathrm{rows}(\mathcal{M})s \\ &\leq (n/\mathrm{cols}(\mathcal{M}) + 1)\,\mathrm{rows}(\mathcal{M})s \\ &= \left(\frac{\mathrm{rows}(\mathcal{M})}{\mathrm{cols}(\mathcal{M})}n + \mathrm{rows}(\mathcal{M})\right)s. \end{aligned}$$

Since $\mathrm{rows}(l) = l\,\mathrm{rows}(\mathcal{M})$ for the construction of the monotone family of incidence matrices and $l = \lceil n/\mathrm{cols}(\mathcal{M}) \rceil \leq n/\mathrm{cols}(\mathcal{M}) + 1$.

Therefore we see that the size of the aggregate signature is linear in n, but the factor $\mathrm{rows}(\mathcal{M})/\mathrm{cols}(\mathcal{M})$ can be made arbitrarily small by choosing a proper cover-free family, such as the one described above based on polynomials over a finite field.

Example Instantiations. Table 4.1 shows parameters of several cover-free families based on the construction described in this section. For each of the rows, there is an instance of our construction that can compress signatures for up to n claims to a vector of m signatures of Σ', while tolerating up to d errors. The numbers q and k are needed for the instantiation of the cover-free family, but do not reflect a property of our construction. Of course, our scheme can be instantiated with different parameters and completely different constructions of cover-free families as well.

q	k	d	$m = \lvert\mathcal{S}\rvert$	$n = \lvert\mathsf{Blocks}\rvert$
5	2	2	25	125
11	2	5	121	1331
17	2	8	289	4913
17	4	4	289	$\approx 1.42 \cdot 10^6$
29	2	14	841	24389
53	2	26	2809	148877
101	2	50	10201	$\approx 1.03 \cdot 10^6$
251	3	83	63001	$\approx 3.97 \cdot 10^9$
1021	2	510	1042441	$\approx 1.06 \cdot 10^9$

Table 4.1: Example parameters for cover-free families.

4.6 Fault-Tolerance for Sequential Schemes

In Section 4.3, a multiset of claim–signature pairs (c_i, τ_i) is said to contain d errors, if d individual signatures τ_i are not regular for their respective claim c_i (see Definition 4.3.5). This definition is not applicable to *sequential* aggregate signatures due to the lack of individual signatures τ_i. Hence, we need to find a different definition to capture fault-tolerance for sequential aggregate signature schemes.

A seemingly natural approach that might come to mind is to define the number of errors via "intermediate" claim sequences $C_i = (c_1, \ldots, c_i)$ and their respective signatures τ_i (this might not be well-defined, but let us ignore this problem for the moment). Following this approach, one might say that a claim sequence C contains d errors if d of the signatures τ_i are not regular outputs of $\mathsf{AggSign}(\mathsf{sk}_t, C_{i-1}, \tau_{i-1}, m_i)$. This approach fails, however, as it does not distinguish between signatures τ_i, which are damaged but sufficiently intact to authenticate some of the claims, and signatures that are completely destroyed. For example, consider the claim sequence $C = (c_1, \ldots, c_n)$ and the signatures $\tau_1, \ldots, \tau_n$, where all τ_i for $i < n$ are regular for the respective intermediate claim sequence $C_i = (c_1, \ldots, c_i)$, but τ_n is completely random. Then there was only one irregular step, and hence only one error with regard to this definition, but $\mathsf{Vfy}(C, \tau_n)$ will output $(\bot, \ldots, \bot)$. Thus, no scheme could even be 1-fault-tolerant, if we would define fault-tolerance in this way.

An alternative way to look at this is to observe that in the definitions for fully flexible aggregate signatures, we assume that the aggregation step is always done correctly, while errors only occur during signing. However, in the sequential aggregate case, these two operations are inseparable, since there is only one algorithm $\mathsf{AggSign}$ for both signing and aggregating and we therefore cannot assume that aggregation did not introduce additional errors.

We therefore restrict our attention to specific changes to the claim sequence C, namely replacements of individual claims as well as the addition or removal of claims *at the end of the sequence*, which we do not consider for fault-tolerant aggregate signatures, since here there is no firmly defined order of aggregation.[9]

For sequential aggregate schemes, however, it seems advantageous to capture these types of "errors" as well, since it more closely follows the $\mathsf{SAS\text{-}EUF\text{-}CMA}$ definition of unforgeability for sequential aggregate signature schemes, which also captures these cases, in contrast to the $\mathsf{AS\text{-}EUF\text{-}CMA}$ definition for aggregate signatures (see Definition 3.2.4 and Definition 3.3.4).

Furthermore, these changes closely model the needed requirements of applications of sequential aggregate signatures like secure logging, as they capture events where an attacker edits log entries or removes tail-end log messages. A more detailed discussion can be found in [Har+17a; Har20].

Definition 4.6.1 (Sequential Aggregate Signature Scheme with List Verification)**.** *A* sequential aggregate signature scheme with list verification *is a tuple* $\Sigma = (\mathsf{Gen}, \mathsf{AggSign}, \mathsf{Vfy})$ *of three PPT algorithms as follows:*

- *The key generation algorithm* $\mathsf{Gen}(1^\kappa)$ *receives the security parameter* κ *as its input and outputs a tuple* $(\mathsf{pk}, \mathsf{sk})$ *consisting of the* public key pk *and the* secret key sk*.*

- *The signature generation and aggregation algorithm* $\mathsf{AggSign}(\mathsf{sk}, C, \sigma, m)$ *receives the secret key* sk*, a claim sequence* C*, a corresponding signature* σ *and a message* m *as its input and outputs a signature* σ^* *for the new claim sequence* $C^* := C \| (\mathsf{pk}, m)$*.*

- *The verification algorithm* $\mathsf{Vfy}(C, \sigma)$ *receives a claim sequence* C *of length* $n \in \mathbb{N}_0$ *and a signature* σ *as input. It outputs a sequence* V *(of length* n*) of claims and error symbols* $\perp$*. We require that for each* $i \in [n]$*, we have that either* $V[i] = C[i]$ *or* $V[i] = \perp$*, i.e.* V *can be obtained from* C *by replacing claims with* $\perp$*. Claims output by* Vfy *are taken to be valid.*

The definition of list verification for sequential schemes differs slightly from the one for fully flexible or synchronized schemes. The Vfy algorithm outputs a sequence with error symbols, since the position of the respective claim in the sequence is important for sequential schemes. For example, the same claim might appear at multiple positions, but might only be valid for some of them. Since no fixed aggregation order exists for fully flexible and synchronized aggregation, the output of their Vfy algorithms is defined more generally, i.e. the output order does not necessarily coincide with the input order, since it might not be possible to deduce the order of aggregation from the aggregate.

[9] Note however that removals of claims *without* also somehow removing their respective "subsignature" from the aggregate signature also results in an error in the definitions for fully flexible aggregate signatures, see Section 5.4 for more details on this.

To capture what we mean by errors for the case of sequential aggregate signatures, we focus on *differences* of claim sequences. For the following definitions, let $C = (c_1, \ldots, c_n)$ and $C' = (c'_1, \ldots, c'_k)$ be two claim sequences.

Definition 4.6.2 (Difference of Claim Sequences)**.** *C and C'* differ on ℓ positions *($0 \leq \ell \leq \min(n, k)$) if $c_i \neq c'_i$ for ℓ indices $1 \leq i \leq \min(n, k)$ and $c_i = c'_i$ for the rest.*

Definition 4.6.3 (Errors in Claim Sequences)**.** *C'* contains d errors with respect to C *if they differ on ℓ positions and $d = |n - k| + \ell$.*

Example 4.6.4. Let c_1, c_2, c_3, c_4 be four distinct claims and let C, C', C^* be three claim sequences with $C = (c_1, c_2, c_3)$, $C' = (c_1, c_2)$ and $C^* = (c_1, c_2, c_4)$.

Then C and C' differ in 0 positions, but C' contains 1 error with respect to C, since $|C| = |C'| + 1$. Analogously, C contains one error with respect to C'. C and C^* differ in one position and C^* contains one error with respect to C.

We can now define fault-tolerance for sequential aggregate signature schemes:

Definition 4.6.5 (Fault-Tolerance for Sequential Aggregate Signature Schemes)**.** *A sequential aggregate signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{AggSign}, \mathsf{Vfy})$ with list verification is* tolerant against d errors*, if for all claim sequences C and C', such that C' contains at most d errors with respect to C and for all signatures τ that are regular for C, we have*

$$\mathsf{Vfy}(C', \tau)[i] = c_i \text{ for all } 1 \leq i \leq \min(n, k) \text{ where } c_i = c'_i.$$

In other words, Vfy *outputs* at least *all claims c_i from C that C' did not modify.*

Definition 4.6.6 (Fault-Tolerant Sequential Aggregate Signature Scheme)**.** *A* d-fault-tolerant sequential aggregate signature scheme *is a sequential aggregate signature scheme with list verification that is tolerant against d errors. A scheme is* fault-tolerant*, if it is d-fault-tolerant for some $d > 0$.*

The generic construction to achieve fault-tolerance for aggregate signatures by using an incidence matrix of a cover-free family from Section 4.5 can also be used to achieve fault-tolerance for sequential aggregate signature schemes by applying it to a sequential scheme. The construction is almost identical to the construction for fully flexible and synchronized schemes, but the $\mathsf{AggSign}$ and Vfy algorithms need to be slightly adapted, so that they respect the sequential nature of aggregation and the modified definition of list verification.

Let $\Sigma' = (\mathsf{Gen}', \mathsf{AggSign}', \mathsf{Vfy}')$ be a sequential aggregate signature scheme with list verification and $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$ a d-cover-free family. Again, signatures of the fault-tolerant scheme $\Sigma_{\mathsf{SeqFT}} = (\mathsf{Gen}_{\mathsf{SeqFT}}, \mathsf{AggSign}_{\mathsf{SeqFT}}, \mathsf{Vfy}_{\mathsf{SeqFT}})$ are vectors of signatures of Σ' and are constructed according to the incidence matrix $\mathcal{M}$ of $\mathcal{F}$. For this purpose, let $C_0 := ()$ and $\tau_0[i] := \lambda$ for each $i \in [\text{rows}(\mathcal{M})]$. Furthermore, if v is a vector, then let $v|_\ell$ denote the vector v, truncated to the first ℓ elements. Then the construction is as follows:

- $\mathsf{Gen}_{\mathsf{SeqFT}}(1^\kappa)$ creates and outputs a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}'(1^\kappa)$.
- $\mathsf{AggSign}_{\mathsf{SeqFT}}(\mathsf{sk}, C_{j-1}, \tau_{j-1}, m_j)$ takes as input a secret key sk, a claim sequence $C_{j-1} = (c_1, \ldots, c_{j-1})$, its corresponding signature τ_{j-1} (which is a vector of signatures of Σ') and a message m_j to sign and aggregate.

The sequential aggregate signature τ_{j-1} is updated component-wise, according to the entries of $\mathcal{M}$: the entries of column j determine in which components of τ_{j-1} the new message m_j gets signed and aggregated.

More precisely, for all $i \in [\text{rows}(\mathcal{M})]$ we set

$$\tau_j[i] := \begin{cases} \mathsf{AggSign}'(\mathsf{sk}, C_{j-1}[\mathcal{M}_i], \tau_{j-1}[i], m_j), & \text{where } \mathcal{M}[i,j] = 1, \\ \tau_{j-1}[i], & \text{where } \mathcal{M}[i,j] = 0. \end{cases}$$

The output is τ_j.

- $\mathsf{Vfy}_{\mathsf{SeqFT}}(C, \tau)$ takes as input a claim sequence C of length $n \in \mathbb{N}_0$ and an aggregate signature τ for C. First, $\mathsf{Vfy}_{\mathsf{SeqFT}}$ computes a bit vector $b \in \{0,1\}^n$ that specifies for each claim if it can safely be considered valid.

  $\mathsf{Vfy}_{\mathsf{SeqFT}}$ initializes b to 0^n and iterates over all entries $\tau[i]$ of τ, letting

  $$b \leftarrow b \vee \mathcal{M}_i|_n \text{ if } \mathsf{Vfy}'(C[\mathcal{M}_i], \tau[i]) = 1$$

  in each iteration. Here, $\vee$ denotes the bitwise logical OR of two bit strings.

  Finally, $\mathsf{Vfy}_{\mathsf{SeqFT}}$ builds the output sequence V component-wise, by letting

  $$V[j] = \begin{cases} C[j], & \text{if } b[j] = 1, \\ \perp, & \text{otherwise,} \end{cases} \qquad \text{for all } j \in [n]$$

  and then outputs V.

Since the proofs for security and fault-tolerance are mostly identical to the proofs for aggregate signature schemes, we only state the following theorems without proving them. Proofs can be found in the full version [Har+17b] of [Har+17a].

Theorem 4.6.7. *If Σ' is a (t', q, ε)-SAS-EUF-CMA secure aggregate signature scheme, then the scheme Σ_{SeqFT} defined above is a (t, q, ε)-SAS-EUF-CMA secure aggregate signature scheme with list verification, where t is approximately the same as t'.*

Theorem 4.6.8. *If $\mathcal{F} = (\mathcal{S}, \mathsf{Blocks})$ is a d-cover-free family, then the sequential aggregate signature scheme Σ_{SeqFT} defined above is tolerant against d errors, and in particular, it is correct.*

Remark. Not all sequential aggregate signature schemes respect the sequential order of aggregation when verifying signatures. For example, the scheme of Lu et al. [Lu+06] does not and here, a signature for a claim sequence C is also valid for any reordering of C (see Section 5.3.6 for a discussion of this scheme). Furthermore, they also use a different definition of SAS-EUF-CMA security than the one given by Lysyanskaya et al. [Lys+04] (also see the discussion below Definition 3.3.4). The Definitions 4.6.2 and 4.6.3 of differences and errors of claim sequences therefore are not very meaningful for these schemes.
One could modify these schemes, so that $\mathsf{AggSign}(\mathsf{sk}, C, \sigma, m)$ aggregates a signature for the message $(|C| + 1)\|\top\|m$, i.e. it now also incorporates the index of m in the current claim sequence into the signature. Here, "$\top$" is some unique symbol that can be used to clearly distinguish the index bit string from the message bit string, so that attackers cannot "move" bits of the index to

the message and vice-versa by clever reorderings and changes to the claims, which could be used to find different claim sequences that are valid for the same signature. When verifying a claim sequence C' with signature σ, the Vfy algorithm will consequentially also need to prepend the indices and $\top$ to the messages. This way, one can enforce that these schemes also need to respect the order of aggregation. Alternatively, for two claim sequences C and C' where, without loss of generality, $|C| \geq |C'|$, we could also say that C contains d errors with respect to C', if $d = |C \setminus C'|$.

For both options, the presented results transfer and in any case, claim sequences in our fault-tolerant construction may not be reordered, since each claim needs to be assigned to a specific and fixed position.

4.7 Application: Robust Secure Logging

In this section we discuss a practical application of fault-tolerance and show how a secure and robust logging scheme can be constructed from a fault-tolerant sequential aggregate scheme.

Remark. This section is taken almost entirely from [Har+17a]. We only give a brief sketch of the results and leave out formal proofs. More details can be found in [Har+17a] and in the PhD thesis of Gunnar Hartung [Har20].

4.7.1 Introduction to Secure Logging

Log files are an indispensable source of information for administrators investigating incidents in a computer system. They provide fine-grained information on actions and events that happened within the system, such as business transactions, errors or security violations. Attackers frequently modify log files to cover their traces, so being able to distinguish real and faked information is crucial.

Therefore, the need to detect modifications to log files is widely recognized among security professionals and much effort has been devoted to finding solutions that unveil such modifications. Cryptographic solutions must be resilient to attackers that gain full control of the log server which holds the secret key. A logging scheme must stay secure *even if the attacker obtains the secret key* at some point in time and it must continue to enable the discovery of illicit log changes which occurred before the secret key was stolen to protect old log entries from unnoticed modification.

Since this is impossible with standard schemes, Anderson [And97] proposed *forward secure* schemes (later formalized in [BM99a], as remarked in [Boy+06]). Here, time is divided into intervals, called *epochs*. For each epoch t, a new secret key is computed from the key of the previous epoch $t-1$. There is only *one single* public key, that can be used to verify signatures created in any epoch, but Vfy also receives an epoch t as its input and signatures are only valid for the epoch t they were created in. By securely erasing old secret keys when they expire, one ensures that an attacker cannot forge signatures for previous epochs.

Detecting *log truncations* (i.e. the removal of log entries at the end) is a surprisingly hard problem, because any authentication information computed by the log server can only authenticate past entries, so there is nothing that authenticates the end of such a chain. Ma and Tsudik [MT09] were the first to

present a mechanism to detect truncations of log files. Their solution is based on forward secure sequential aggregate signatures.

The core property of their solution lies in the fact that unforgeability for sequential aggregate signature schemes implies that removing a message from a given sequential aggregate signature is intractable in a restricted sense (we discuss this topic of *deaggregation* further in Chapter 5). Ma and Tsudik [MT09] then use this property by storing only a single signature for the entire log file, which is an aggregate of the signatures for each individual message. The supposed hardness of removing an individual signature then guarantees that no attacker can remove *any* message from the log file without notice, as in truncation attacks. However, their approach is not *robust*, i.e. a single erroneous log entry renders the signature invalid, without any information on where the error is located. In a following investigation, distinguishing between real and faked information is no longer possible.

Only two solutions proposed so far are robust and truncation-secure at the same time, namely the second immutable scheme in [MT09], and the scheme of Hartung [Har16]. Unfortunately, this robustness property is "bought" by falling back to individual signatures for each log entry, resulting in a very large log signature that is linear in the log size. Moreover, the truncation security of the scheme of [MT09] is only argued informally without a rigorous proof.

By combining fault-tolerance for sequential aggregate signature schemes and forward security, we can construct a provably secure and robust logging scheme that is publicly-verifiable (as defined in [Hol06]), and features short signatures, robustness and truncation resistance. The security reduction is tight and the scheme does not require public ledgers (e.g. blockchains) or any other third party that needs to vouch for the integrity of the log file.

4.7.2 Forward Security

A forward secure digital signature scheme [BM99a] uses distinct secret keys for signing for different time intervals called *epochs*. The main idea is to provide security for previous epochs, even when the secret key of the current epoch gets compromised in some way (for example even if an attacker would gain full knowledge of the secret key).

Throughout this section we assume without loss of generality that the current epoch number can be efficiently derived from the current secret key. The verification algorithm also receives the epoch number as input and a signature computed for a specific epoch t is only valid for this epoch number. Therefore, claims now also store the respective epoch number.

All secret keys offer the same functionality, i.e. all signatures computed under any of theses keys are valid for the same public key. Forward secure schemes then offer an additional algorithm Update, which given the secret key of epoch t, efficiently computes a key of the next epoch $t+1$. We call schemes that offer this functionality *key-evolving*. Furthermore, it should be computational infeasible to compute any secret key of any previous epoch or to forge any signature valid for a previous epoch, even if the secret key of the current epoch is known. This way, all signatures for epochs before the one in which the secret key was compromised remain functional and still securely authenticate their respective messages. These properties are then captured by the security definitions for key-evolving schemes. As an example, we now formally define these concepts for digital signatures.

Definition 4.7.1 (Key-Evolving Digital Signature)**.** *A* key-evolving digital signature signature *scheme is a tuple of PPT algorithms* $\Sigma = (\mathsf{Gen}, \mathsf{Update}, \mathsf{Sign}, \mathsf{Vfy})$*, where*

- $\mathsf{Gen}(1^\kappa, 1^T)$ *takes as input the security parameter* κ *and an a priori upper bound* T *on the number of epochs. It outputs a key pair* $(\mathsf{pk}, \mathsf{sk}_0)$*, where* sk_0 *is the secret key for the first epoch.*
- $\mathsf{Update}(\mathsf{sk}_t)$ *takes as input the secret key* sk_t *of period* t*. If* $t \geq T-1$ *its output is not defined. If* $t < T-1$ *it computes the secret key* sk_{t+1} *for the following period* $t+1$ *and securely erases the old secret key* sk_t*.*
- $\mathsf{Sign}(\mathsf{sk}_t, m)$ *takes as input a secret key* sk_t *and a message* $m \in \{0,1\}^*$ *and outputs a signature* σ *for claim* (pk, t, m)*, where* t *is the epoch of* sk_t*.*
- $\mathsf{Vfy}((\mathsf{pk}, t, m), \sigma)$ *outputs* 1 *if* σ *is a valid signature for the message* m *in epoch* t *under public key* pk*, and* 0 *otherwise.*

Definition 4.7.2 (Correctness for Key-Evolving Digital Signature Schemes)**.** *A key-evolving digital signature scheme is* correct *if for all epoch bounds* $T = \mathrm{poly}(\kappa)$*, all indices* $t \in \{0, \ldots, T-1\}$*, all messages* $m \in \{0,1\}^*$ *and all* $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\kappa, 1^T)$ *and* $\mathsf{sk}_{t+1} = \mathsf{Update}(\mathsf{sk}_t)$ *for* $t = \{0, \ldots, T-2\}$*, it holds that*

$$\mathsf{Vfy}((\mathsf{pk}, t, m), \mathsf{Sign}(\mathsf{sk}_t, m)) = 1.$$

Definition 4.7.3 (FS-EUF-CMA Security for Key-Evolving Digital Signature Schemes)**.** *The security experiment for key-evolving digital signatures schemes consists of four phases and is based on [BM99a]. The general idea is that an attacker should not be able to forge a signature for any earlier epoch, even if he knows the secret key of the current epoch.*

Setup Phase. *The challenger* $\mathcal{C}$ *generates a key pair* $(\mathsf{pk}^*, \mathsf{sk}_0^*) \leftarrow \mathsf{Gen}(1^\kappa, 1^T)$ *(where* T *is the maximal number of epochs) and gives the public key* pk^* *and* T *to the attacker. It sets* $t := 0$*.*

Query Phase. *The attacker* $\mathcal{A}$ *has access to an* Update *and a* Sign *oracle. When* $\mathcal{A}$ *calls the* Update *oracle,* $\mathcal{C}$ *computes* $\mathsf{sk}_{t+1}^* := \mathsf{Update}(\mathsf{sk}_t^*)$*, sets* $t := t+1$*, and returns "ok".* $\mathcal{A}$ *may only make* $T-1$ Update *queries.* $\mathcal{A}$ *may (adaptively) issue signature queries to the* Sign *oracle for messages* m *of his choosing. For these queries, the challenger responds with a signature* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_t^*, m)$*.*

Break In Phase. $\mathcal{A}$ *may send a* break in *request to obtain the current secret key.* $\mathcal{C}$ *sets* $t_{\mathsf{BreakIn}} := t$ *and sends* sk_t *to* $\mathcal{A}$*. Afterwards,* $\mathcal{A}$ *is denied any further access to the oracles. We set* $t_{\mathsf{BreakIn}} := \infty$ *if* $\mathcal{A}$ *does not break in.*

Forgery Phase. *Finally,* $\mathcal{A}$ *outputs a claim* $(\mathsf{pk}^*, t^*, m^*)$ *and a corresponding signature* σ^**.*

The attacker $\mathcal{A}$ wins *the experiment if* σ^* *is a valid signature for the claim* $(\mathsf{pk}^*, t^*, m^*)$*,* m^* *was not queried to the* Sign *oracle during period* t^**, and* $t^* < t_{\mathsf{BreakIn}}$*.*

A key-evolving digital signature scheme is FS-EUF-CMA *secure*[10] *if for each* $T = \text{poly}(\kappa)$ *the success probability in winning the above experiment is negligible in the security parameter* κ *for all PPT attackers* $\mathcal{A}$*.*

The definitions of fully flexible, sequential and synchronized aggregate schemes can be easily reformulated in the same way to also encompass forward-security. Since these reformulations are straightforward, we do not formally define them here. Formal definitions for sequential schemes can be found in [Har+17a]. The prefix "FS" (standing for "Forward Secure") is used to distinguish the forward secure definitions from the common definitions.

4.7.3 Formal Definition of Robust Secure Logging

We now present the formal definition of robust and secure logging from [Har+17a].

Definition 4.7.4 (Logging Scheme with List Verification)**.** *A* logging scheme with list verification $\Lambda = (\mathsf{Gen}, \mathsf{Append}, \mathsf{Update}, \mathsf{ValidEntries}, \mathsf{VfyLog})$ *is a tuple of five PPT algorithms, where*

- $\mathsf{Gen}(1^\kappa, 1^T)$ *takes as input the security parameter* κ *and an a priori upper bound* T *on the number of epochs. It outputs a key pair* $(\mathsf{pk}, \mathsf{sk}_0)$*, where* sk_0 *is the secret key for the first epoch.*

- $\mathsf{Append}(\mathsf{sk}_t, C_{i-1}, \sigma_{i-1}, m_i)$ *takes as input a secret key* sk_t *for epoch* t*, a claim sequence* C_{i-1}*, a corresponding signature* σ_{i-1} *and a message* m_i*. It outputs a signature* σ_i *for the new claim sequence* $C_i := C_{i-1} \| (\mathsf{pk}, t, m_i)$*, thereby adding* m_i *to the log.*[11]

- $\mathsf{Update}(\mathsf{sk}_t, C, \sigma)$ *takes as input the secret key* sk_t *of period* t*. If* $t \geq T - 1$ *the output is undefined. If* $t < T - 1$ *it computes the secret key* sk_{t+1} *for period* $t + 1$ *and* securely erases *the old key* sk_t*. The claim sequence* C *and signature* σ *may be modified, e.g. to add epoch markers [BY97].*

- $\mathsf{ValidEntries}(C, \sigma)$ *takes as input a claim sequence* C *of length* $n \in \mathbb{N}_0$ *and a signature* σ *for* C *and outputs a sequence* V *(also of length* n*) of claims and error symbols* $\perp$*. We require that for each* $i \in [n]$*, either* $V[i] = C[i]$ *or* $V[i] = \perp$ *(i.e.* V *can be obtained from* C *by replacing claims with* $\perp$*). Claims output by* Vfy *are taken to be valid.*

- $\mathsf{VfyLog}(C, \sigma)$ *outputs either* $\emptyset$*, if the signature is without errors, or a subset of a set of error symbols* $\mathcal{E}$*, otherwise. We set* $\mathcal{E} := \{\perp_{\text{sig}}, \perp_{\text{len}}, \perp_{\text{em}}\}$*, with the interpretation that* $\perp_{\text{sig}} \in \mathsf{VfyLog}(C, \sigma)$ *if the signature is not valid, i.e.* $\mathsf{ValidEntries}(C, \tau) \neq C$*. Moreover, if* $\perp_{\text{len}} \in \mathsf{VfyLog}(C, \sigma)$*, the signature may have been truncated. Finally,* $\perp_{\text{em}} \in \mathsf{VfyLog}(C, \sigma)$ *if some problem with epoch markers has been detected.*

Fault-tolerance is defined analogously to the definition for sequential aggregate signature schemes (see Section 4.6), substituting Append for AggSign, and

[10] FS-EUF-CMA stands for *Forward Secure Existentially Unforgeable Under Chosen Message Attacks.*

[11] For efficiency, if the log-file is used with only one public key, it suffices to add it once, instead of adding it to each log entry.

ValidEntries for Vfy. A logging scheme with list verification is *robust* if it is fault-tolerant and we have that regular log files, meaning log files created by correctly using the algorithms of the log scheme, are *error-free* (i.e. $\mathsf{VfyLog}(C, \sigma) = \emptyset$) and error-free log files are valid (i.e. $\mathsf{ValidEntries}(C, \tau) = C$). Note that if the signature is valid in the sense that all claims are returned by ValidEntries, it is still possible that an attacker might have truncated the log. In this case an error symbol returned by VfyLog points towards this possibility.

The following security notion for logging schemes originally presented in [Har+17a] is similar to unforgeability for forward secure sequential aggregate signature schemes, but models the real-world setting of secure logging more closely: The log server maintains a state which the attacker influences only through his oracles. In more detail, a log append oracle appends an entry to the internal log file and an attacker can never add messages to any earlier state of the log file. Moreover, the internal signatures remain hidden from the attacker by default, as these usually stay on the server.

To strengthen the notion, attackers have access to an additional oracle returning the current signature, which models a public verification of the log file by a third party. To exclude trivial attacks, attackers are not allowed to truncate the log file to a state they queried a signature for. However, attackers may try to, for example, use these signatures to truncate the log file to a different previous state.

At the end of the experiment, the attacker outputs a forgery. Furthermore, error-freeness is required, i.e. $\mathsf{VfyLog}(C^*, \sigma^*) = \emptyset$, as otherwise the attacker might use a combination of faults and truncations of the claim sequence to obtain a valid signature (i.e. verification of the forged signature and claim sequence outputs the full forged claim sequence) that violates other anti-truncation mechanisms.

Definition 4.7.5 (FS-EUF-CLMA Security for Logging Schemes with List Verification)**.** *For a log scheme with list verification* $\Lambda = (\mathsf{Gen}, \mathsf{Append}, \mathsf{Update}, \mathsf{ValidEntries}, \mathsf{VfyLog})$*, a PPT attacker* $\mathcal{A}$*, the number of epochs* T *and the security parameter* $\kappa \in \mathbb{N}_0$*, the* FS-EUF-CLMA[12] *security experiment is defined as follows:*

Setup Phase. *The experiment generates a key pair* $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\kappa, 1^T)$*, the log file* $C_0 := ()$ *and signature* $\sigma_0 := \lambda$*. It initializes the epoch counter* $t := 0$*, and starts* $\mathcal{A}$ *with inputs* pk*,* T*.*

Query Phase. $\mathcal{A}$ *may adaptively issue queries to the following oracles:*

- **LogAppend Oracle.** *The experiment appends the specified message* m *to the log and updates the signature via* $\sigma_i \leftarrow \mathsf{Append}(\mathsf{sk}_t, C_{i-1}, \sigma_{i-1}, m)$*, where* σ_{i-1} *denotes the previous signature, and returns "ok".*
- **NextEpoch Oracle.** *The oracle updates the secret key, the log and its signature via* $\mathsf{Update}(\mathsf{sk}_t, C_{i-1}, \sigma_{i-1})$*, increments the epoch counter* $t := t + 1$ *and returns "ok". It may be queried at most* $T - 1$ *times.*
- **GetSignature Oracle.** *Whenever* $\mathcal{A}$ *calls the* GetSignature *oracle, the challenger responds with the current signature* σ_i *of the log.*

[12] FS-EUF-CLMA stands for *Forward Secure Existentially Unforgeable Under Chosen Log Message Attacks.*

Break In Phase. *The attacker may break in to obtain the current secret key* sk_t*. If* $\mathcal{A}$ *does, the experiment sets* $t_{\mathsf{BreakIn}} := t$*. Otherwise, let* $t_{\mathsf{BreakIn}} := \infty$*.*

Forgery Phase. $\mathcal{A}$ *outputs a log file* C^**, and a forged signature* σ^* *for* C^**.*

$\mathcal{A}$ wins *the experiment, if the following conditions hold:*

- *The signature* σ^* *is* error-free*, i.e.* $\mathsf{VfyLog}(C^*, \sigma^*) = \emptyset$*, which also implies that the signature is valid.*
- *The signature is* non-trivial*, meaning the following: Let* C' *be the subsequence of* C^* *that is obtained by deleting all claims* $c = (\mathsf{pk}, t, m)$ *from* C^**, where* $t \geq t_{\mathsf{BreakIn}}$*. The forgery is* non-trivial*, if and only if* $|C'| \neq 0$ *and* C' *does not equal the content of the log file during any* GetSignature *query.*

A logging scheme with list verification Λ *is said to be* FS-EUF-CLMA secure*, if and only if for all* $T = T(\kappa) \in \mathrm{poly}(\kappa)$ *and all PPT attackers* $\mathcal{A}$ *the probability for* $\mathcal{A}$ *winning the above experiment is negligible in the security parameter* κ*.*

4.7.4 Construction of a Robust Secure Logging Scheme

We now present our generic construction of a secure and robust log scheme $\Lambda = (\mathsf{Gen}, \mathsf{Append}, \mathsf{Update}, \mathsf{ValidEntries}, \mathsf{VfyLog})$ using a fault-tolerant sequential aggregate signature.

Let $\mathsf{SAS} = (\mathsf{Gen}_{\mathsf{SAS}}, \mathsf{Update}_{\mathsf{SAS}}, \mathsf{AggSign}_{\mathsf{SAS}}, \mathsf{Vfy}_{\mathsf{SAS}})$ be a key-evolving sequential aggregate scheme with list verification and $\mathsf{FS} = (\mathsf{Gen}_{\mathsf{FS}}, \mathsf{Update}_{\mathsf{FS}}, \mathsf{Sign}_{\mathsf{FS}}, \mathsf{Vfy}_{\mathsf{FS}})$ a key-evolving digital signature scheme.

- $\mathsf{Gen}(1^\kappa, 1^T)$ creates key pairs of SAS and FS as

$$(\mathsf{pk}_{\mathsf{SAS}}, \mathsf{sk}_{\mathsf{SAS}}) \leftarrow \mathsf{Gen}_{\mathsf{SAS}}(1^\kappa, 1^T),$$
$$(\mathsf{pk}_{\mathsf{FS}}, \mathsf{sk}_{\mathsf{FS}}) \leftarrow \mathsf{Gen}_{\mathsf{FS}}(1^\kappa, 1^T)$$

  and returns $\mathsf{pk} = (\mathsf{pk}_{\mathsf{SAS}}, \mathsf{pk}_{\mathsf{FS}})$ and $\mathsf{sk}_0 = (\mathsf{sk}_{\mathsf{SAS}}, \mathsf{sk}_{\mathsf{FS}})$.

- $\mathsf{Append}(\mathsf{sk}_t, C_{i-1}, \tau_{i-1}, m_i)$ takes as input a secret key $\mathsf{sk}_t = (\mathsf{sk}_{\mathsf{SAS}}, \mathsf{sk}_{\mathsf{FS}})$ for period t, a claim sequence $C_{i-1} = (c_1, \ldots, c_{i-1})$, its corresponding signature $\tau_{i-1} = (\sigma_{i-1}, s_{i-1})$ and a message m_i to sign. It uses the signing algorithms of SAS and FS to compute a signature on the entries of the log and a signature on the length of the log via

$$\sigma_i \leftarrow \mathsf{AggSign}_{\mathsf{SAS}}(\mathsf{sk}_{\mathsf{SAS}}, C_{i-1}, \sigma_{i-1}, m_i \,\|\, i), \text{ and}$$
$$s_i \leftarrow \mathsf{Sign}_{\mathsf{FS}}(\mathsf{sk}_{\mathsf{FS}}, i).$$

  Append securely erases the old length signature s_{i-1} and the old log signature σ_{i-1}, so that they cannot be used in case of a later break in. The resulting signature $\tau_i = (\sigma_i, s_i)$ is returned.

- $\mathsf{Update}(\mathsf{sk}_t, C_{i-1}, \tau_{i-1})$ takes as input the secret key $\mathsf{sk}_t = (\mathsf{sk}_{\mathsf{SAS}}, \mathsf{sk}_{\mathsf{FS}})$, a claim sequence C_{i-1} and a corresponding signature τ_{i-1}. It appends an *epoch marker* to the log file that is valid for the current epoch t via

$$\tau_i \leftarrow \mathsf{Append}(\mathsf{sk}_t, C_{i-1}, \tau_{i-1}, \texttt{"End of epoch:"} \,\|\, t).$$

It then updates the components of sk_t via $\mathsf{sk}'_{\mathsf{SAS}} \leftarrow \mathsf{Update}_{\mathsf{SAS}}(\mathsf{sk}_{\mathsf{SAS}})$ and $\mathsf{sk}'_{\mathsf{FS}} \leftarrow \mathsf{Update}_{\mathsf{FS}}(\mathsf{sk}_{\mathsf{FS}})$. These algorithms also erase the old keys securely. The new secret key is $\mathsf{sk}_{t+1} = (\mathsf{sk}'_{\mathsf{SAS}}, \mathsf{sk}'_{\mathsf{FS}})$, the new claim sequence is $C_i = C_{i-1} \,\|\, (\mathsf{pk}, t, \texttt{"End of epoch:"} \,\|\, t)$, and the new signature is τ_i.

- $\mathsf{ValidEntries}(C, \tau)$ takes as input a claim sequence C and a signature $\tau = (\sigma, s)$ for C. It outputs $\mathsf{Vfy}_{\mathsf{SAS}}(C', \sigma)$, where C' is the claim sequence generated from C by appending the message number i to m_i for all claims in C.

- $\mathsf{VfyLog}(C, \tau)$ takes as input a claim sequence C and a signature $\tau = (\sigma, s)$ for C. It maintains an error set E initialized to $\emptyset$.

  First, it verifies the FS signature s by computing $b = \mathsf{Vfy}_{\mathsf{FS}}((\mathsf{pk}_{\mathsf{FS}}, t, |C|), s)$. If $b = 0$, it adds $\perp_{\text{len}}$ to E.

  Then it proceeds with checking the epoch markers: For all claims $c_i = (\mathsf{pk}, t_i, m_i)$ and $c_{i+1} = (\mathsf{pk}, t_{i+1}, m_{i+1})$ in C, where $t_{i+1} \neq t_i$, it considers two cases. If $t_{i+1} \neq t_i + 1$ then it adds $\perp_{\text{em}}$ to E, else it check ifs

  $$m_i = \texttt{"End of epoch:"} \,\|\, t_i.$$

  If not, it adds $\perp_{\text{em}}$ to E.

  Next, it checks whether the signature is valid, i.e. $\mathsf{ValidEntries}(C, \tau) = C$, and adds $\perp_{\text{sig}}$ to E, if this is not the case. Finally, it outputs the set of errors E.

The construction fulfills the desired definitions of security and fault-tolerance and has a tight security proof:

Theorem 4.7.6. *The log scheme Λ described above is d-fault-tolerant, if the used key-evolving sequential aggregate signature scheme* SAS *is d-fault-tolerant.*

Proof. A proof of this theorem can be found in the full version [Har+17b] of [Har+17a]. □

Theorem 4.7.7. *The log scheme Λ described above is* FS-EUF-CLMA *secure, if* SAS *is* FS-SAS-EUF-CMA *secure and* FS *is* FS-EUF-CMA *secure.*

More precisely, for any PPT adversary $\mathcal{A}$ who breaks the FS-EUF-CLMA *security with success probability $\varepsilon_{\mathcal{A}}$, there exists a PPT adversary $\mathcal{B}$ who either breaks the* FS-SAS-EUF-CMA *security of* SAS *or the* FS-EUF-CMA *security of* FS *with success probability at least $\varepsilon^{\mathsf{SAS}}_{\mathcal{B}} \geq \frac{\varepsilon_{\mathcal{A}}}{2}$ and $\varepsilon^{\mathsf{FS}}_{\mathcal{B}} \geq \frac{\varepsilon_{\mathcal{A}}}{2}$, respectively.*

Proof. A proof of this theorem can be found in [Har+17a]. □

Practical instantiations of the construction can be obtained by using the schemes of [BM99a; MT07; Ma08].[13] See Section 5 of [Har+17a] for a detailed discussion of its performance.

[13] Note that several of the forward secure sequential aggregate schemes proposed by Ma [Ma08] were later shown to be insecure by Hartung [Har17].

Chapter 5

Deaggregation Security

5.1 Introduction

In this chapter we discuss the second main contribution of this thesis, which is *deaggregation security* of aggregate signature schemes. The question is if attackers can remove claims from a given aggregate signature and through this compute signatures for claim sequences that were never honestly signed. This would enable the attacker to selectively remove unwanted messages and this type of attack could, for example, be used to remove entries of a database that is protected by an aggregate signature scheme.

To illustrate this, suppose σ is an aggregate signature for three distinct claims c_1, c_2 and c_3 of three different signers s_1, s_2 and s_3. Then an attacker that knows σ_{Agg} might be able compute new signatures that are valid for some subset of these claims, by removing or *deaggregating* individual signatures that were used to compute σ. In the example, the attacker might be able to compute a signature valid for c_1 and c_2 from σ_{Agg}, even if no such signature was ever computed or made public by the signers.

While aggregation generally is a public operation and no secret information is needed to aggregate, signers are not required to publicly release individual signatures for their messages. So, the signers could have cooperated to secretly compute σ_{Agg} and then may have released it without revealing any of the intermediate signatures. Another scenario could be that s_1 and s_3 released individual signatures for c_1 and c_3, which s_2 then aggregated together with a signature σ_2 for c_2 to compute σ_{Agg}, without ever publicly revealing σ_2. In both cases, it should not be possible for an attacker to compute a signature that is valid for c_1 and c_2 from σ_{Agg}.

At first glance, one might assume that aggregation is an irreversible commitment step and therefore such attacks should not be possible. If this would be true, aggregate signatures would also provide protection against unwanted deletion of data and the retraction of signatures, which could have a wide range of applications, from secure logging [MT09], securely storing databases, constructing other cryptographic primitives like verifably encrypted signatures [Bon+03] to improving cryptocurrencies [SMD14]. Unfortunately, this intuition turns out to be false and aggregation is often a reversible operation.

Many schemes operate in groups, signatures are group elements and aggregation is done by simply multiplying these elements. Therefore, if an attacker knows a signature σ for a claim sequence C and another signature σ' for a subsequence $C' \subsetneq C$, she can simply invert σ' and *deaggregate* the corresponding claims by computing $\sigma \cdot \sigma'^{-1}$, thereby creating a signature for the claims $C \setminus C'$. For example, the schemes of Boneh et al. [Bon+03] and Ahn et al. [AGH10] can be exploited this way (see Section 5.3.1 and Section 5.3.7 for a detailed discussion of their deaggregation security).

Furthermore, unforgeability also does not imply deaggregation security, since the signature resulting from such a deaggregation attack might *not* contain a forgery in the sense of unforgeability. Suppose an attacker observed a signature σ_{Agg} for three distinct claims c_1, c_2, c_3 and an individual signature σ_2 for c_2, these are all signatures that were ever computed by the signers and the attacker knows none of the secret keys. Now, we would expect that the attacker cannot compute a signature σ' valid for the claims c_1 and c_3 by somehow modifying σ_{Agg} and σ_2, since the signers never issued such a signature. However, since this signature contains no new claims for any of the keys, it does *not* represent a valid forgery. Therefore this type of attack is not ruled out by unforgeability definitions. In fact, many schemes are susceptible to such attacks (again, for example the schemes of [Bon+03; AGH10]), although their unforgeability was formally proven and it would be possible to compute a signature for c_1 and c_3 from σ_{Agg} and σ_2 in the example above. We discuss the relationship between unforgeability and deaggregation security in more detail in Section 5.2.4.

The goal of this chapter is to clearly define and discuss deaggregation security, as well as to analyze the potential deaggregation security of several existing schemes.

5.1.1 Contribution

We formally define several definitions that capture security against deaggregation attacks in various strengths. We discuss the relationships between these security definitions, the definition of Saxena et al. [SMD14] and unforgeability and thereby show that there exists a formal hierarchy of deaggregation security definitions. For aggregate signature schemes that exhibit three rather natural properties, we also show that several levels of this hierarchy collapse (i.e. basic definitions imply stronger ones). Furthermore, we discuss the deaggregation (in-) security of a range of aggregate signature schemes published in recent years and precisely classify their offered deaggregation security by giving new deaggregation security proofs and attacks and thereby formally show which definition of the hierarchy they fulfill. Surprisingly, almost all schemes discussed offer at least a basic protection against deaggregation attacks. Unfortunately, no scheme fulfills the strong definition of Saxena et al., except for their own scheme, which is a variant of the scheme of Boneh et al. [Bon+03], is only secure in the random oracle model and impractical for many applications, since the signature size grows *linearly* in the number of claims.

We also investigate the connection between fault-tolerance and deaggregation security. Unfortunately, we can show that it is impossible for aggregate signature schemes to be fault-tolerant and deaggregation secure at the same time. In fact, the argumentation is surprisingly straightforward: Let σ_{Agg} be an aggregate signature for a claim sequence C of a scheme that can at least tolerate one error.

For simplicity, assume that σ_{Agg} contains no errors yet. A deaggregation attacker then only needs to remove a claim c^* from C *without* modifying σ_{Agg}. Now, according to the definitions of fault-tolerant schemes, the Vfy algorithm will output $C \setminus (c^*)$, i.e. the attacker has successfully removed a claim by introducing one fault. We argue this more formally in Section 5.4.

This unfortunately implies that for applications of aggregate signatures, a trade-off between fault-tolerance and deaggregation security needs to be made. One needs to carefully decide which of the two properties should be provided directly by the aggregation scheme itself and which property needs to be protected by additional security measures such as added back-up mechanisms or other cryptographic primitives added on top of the aggregate signature scheme.

Remark. Minor parts of this chapter are based on work done together with Roland Gröll and were already published in his diploma thesis [Grö16] that was supervised by the author of this thesis. However, all parts that also appear in [Grö16] are greatly improved, expanded and generalized and most of the results presented here do not appear in [Grö16].

5.1.2 Related Work

Boneh, Gentry, Lynn and Shacham already discussed a very basic notion of deaggregation security in their seminal work that introduced the concept of aggregate signatures [Bon+03] and show that it has interesting applications. Their notion basically states that given *one single* aggregate signature (and no other signatures or access to a signing oracle), it is hard to remove one of the aggregated claims. Interestingly, they do not formulate their notion as a formal security definition for aggregate signature schemes, but rather as a computational problem on groups (since their signatures are group elements) and also do not give formal justifications why they assume that this problem is computationally hard. They then use this assumption to construct a verifiably encrypted signature scheme from their aggregate signature scheme. Fortunately, Coron and Naccache [CN03] later showed that this computational problem is equivalent to the Computational Diffie-Hellman problem. This problem, called the *k-Element Aggregate Extraction Problem*, is also discussed in Section 5.2.1.

The research on aggregate signatures that followed the work of Boneh et al. was mostly focused on constructing new aggregate signature schemes and did not pay much attention to deaggregation security. To the best of our knowledge, most of the aggregate signature schemes published after the work of Boneh et al. were not constructed with deaggregation security in mind and were never analyzed from this viewpoint.

However, sequential aggregate signatures are a special case. Here, the definition of unforgeability of Lysyanskaya et al. [Lys+04] for sequential schemes (called $\mathsf{SAS\text{-}EUF\text{-}CMA}$ security in this thesis, see Definition 3.3.4) already implies a very restricted form of deaggregation security. In unforgeability definitions, the forgery of the attacker usually must contain at least one message signed under the challenge public key pk^* that the attacker never queried a signature for. $\mathsf{SAS\text{-}EUF\text{-}CMA}$ security softens this restriction: Suppose the attacker sends a signature σ' for a claim sequence C and asks the challenger to aggregate a signature for m^* to σ'. The challenger will then answer with a signature σ_{Agg} on the claim sequence $C \| (\mathsf{pk}^*, m^*)$. Now, the attacker would also win the $\mathsf{SAS\text{-}EUF\text{-}CMA}$

experiment if he could output a signature valid for a claim sequence $C' \| (\mathsf{pk}^*, m^*)$ with $C' \neq C$, even if C' contains no new claim using pk^*. Therefore, the attacker would win the experiment, if he could change the "prefix" C of his query. Such a change can be a reordering of the claims in the prefix C, a modification of one of the messages, inserting a new claim or, in fact, a deaggregation of a claim of the prefix. Surprisingly, this type of unforgeability still does not imply any form of general deaggregation security, as we discuss in Section 5.2.4. In fact, the scheme of Lysyanskaya et al. [Lys+04] itself is susceptible to deaggregation attacks (see and Section 5.3.4). Still, Ma and Tsudik [MT09] state that they use this property of sequential schemes to construct a secure logging scheme. However, they do not provide a rigorous proof for their construction and also do not formally define which type of deaggregation security they exactly assume. Furthermore, not all sequential schemes fulfill this definition of unforgeability, for example the schemes of Lu et al. [Lu+06] and Lee, Lee, and Yung [LLY13b; LLY13a] use a different definition where such forgeries are not seen as valid.

Mykletun, Narasimha, and Tsudik [MNT04] consider the concept of *immutable* aggregate signatures in the context of database outsourcing. Here, the idea is that aggregate signatures cannot be modified *at all* without some secret knowledge (meaning aggregation is no longer a public operation), which also seems to imply deaggregation security. Unfortunately, like [MT09] they also give no formal definition of immutability and do not formally prove the security of their schemes. Furthermore, they assume the existence of a trusted server, which is also the only party able to aggregate.

Fischlin, Lehmann, and Schröder [FLS12] discuss *history-freeness* of sequential aggregate signatures. Here, the $\mathsf{AggSign}$ algorithm does not receive the messages and public keys so far as input to improve performance. The attacker might now use partial signatures in its signature queries that are invalid or for which he does not know the corresponding messages. Because the $\mathsf{AggSign}$ algorithm no longer receives the necessary information to verify the signature, this might give the attacker an advantage in breaking the scheme. In fact, most sequential schemes critically rely on this verification step in the $\mathsf{AggSign}$ algorithm for their security [BGR14]. Therefore, security needs to be defined differently and they give two new definitions that also encompass a strong form of deaggregation security. However, as already stated by Fischlin et al. themselves, it is unclear whether their strong notion can be fulfilled. For their relaxed notion, they present a scheme based on [Bon+03], which is secure in the random oracle model. However, their definitions do not transfer to fully flexible or synchronized aggregation, since they can only be achieved by schemes where aggregation is not a publicly computable operation.

Saxena, Misra, and Dhar [SMD14] present a very strong notion of deaggregation security and show how it can be applied to increase the anonymity of the Bitcoin cryptocurrency [Nak09]. They also construct a scheme that fulfills their definition. Their construction is a variant of the scheme of Boneh et al. and has the serious drawback that the size of aggregate signatures grows *linearly* in the number of claims. Furthermore, their notion is tailor-made for their scheme and several aspects of it need to be generalized so that it can also be applied to other schemes as well (see Section 5.2.7).

5.1.3 Overview

Section 5.2 presents and discusses a hierarchy of different security definitions that capture security against deaggregation attacks. Section 5.2.2 and Section 5.2.3 present basic definitions called nDEAGG and nDEAGG$^+$ security, that only offer limited protection, but are nonetheless interesting from both a theoretical and practical standpoint. Here, the attacker may query only *one single* aggregate signature for a claim sequence of n claims (and has no access to a general signing oracle) and then needs to deaggregate at least one claim to win.

Section 5.2.4 discusses the connection between deaggregation security and unforgeability and shows that unforgeability does not imply any form of deaggregation security. Section 5.2.5 discusses the relationship between different levels of nDEAGG and nDEAGG$^+$ security for different n. The main result is that for schemes that exhibit three natural properties (called extendability, claim-removability and order-independence) 2DEAGG, respectively 2DEAGG$^+$, security implies nDEAGG, respectively nDEAGG$^+$ security, for $n \geq 2$. Section 5.2.6 then discusses these security definitions in detail.

In Section 5.2.7 a very strong definition capturing a high level of deaggregation security given by [SMD14], which we call ADEAGG security, is introduced, generalized and discussed. To conclude this section of security definitions for deaggregation security, Section 5.2.9 gives an overview over the different security definitions and their relationships.

Section 5.3 then discusses the deaggregation (in-) security of several known aggregate signature schemes, namely the schemes of [Bon$^+$03; Lys$^+$04; Lu$^+$06; BNN07; Nev08; AGH10; SMD14] and presents formal deaggregation security proofs and new attacks on these schemes.

Finally, in Section 5.4 we discuss the connection between deaggregation security and fault-tolerance and show that no fault-tolerant scheme can be deaggregation secure.

5.2 Definitions of Deaggregation Security

In this section, we introduce several security definitions that capture deaggregation security. We present a hierarchy of different definitions that successively capture stronger forms of deaggregation security. The first definitions (nDEAGG and nDEAGG$^+$ security) are inspired by the k-Element Aggregate Extraction assumption of [Bon$^+$03]. They only capture a basic level of deaggregation security and only provide very limited protection, since they allow the attacker only *one* signature query for a claim sequence of length n.

However, they are still interesting from both a theoretical and practical standpoint (see Section 5.2.6 for a detailed discussion). Also, most known schemes fulfill only these basic definitions and are susceptible to stronger deaggregation attacks, as we show in Section 5.3.

The strongest definition presented in this thesis, called ADEAGG security, is a generalized form of the definition given by [SMD14] that succinctly captures a high level of security and allows the attacker to adaptively ask for signatures. Unfortunately, there exists no practical scheme that fulfills it. This section also discusses the relationships between these definitions and their connection to unforgeability.

5.2.1 The k-Element Aggregate Extraction Assumption

As already mentioned in the section on related work, Boneh et al. [Bon+03] already introduced the *k-Element Aggregate Extraction Problem* and the associated *k-Element Aggregate Extraction Assumption*, which captures a weak form of deaggregation security for their scheme. The assumption is not formulated as a security property of aggregate signatures, but rather as a computational problem on algebraic groups. It is introduced in a general form for an unspecified value of k, but since we only use the assumption for the special case of $k = 2$ in this thesis, we only present it for this case:

Definition 5.2.1 (2-Element Aggregate Extraction Assumption)**.** *Let $\mathbb{G}$ be a cyclic group of prime order p and g a random generator of $\mathbb{G}$ depended on the security parameter κ. Let $a, b, u, v \leftarrow \mathbb{Z}_p$. The* 2-Element Aggregate Extraction Assumption *(abbreviated as* 2EAE*) states that for all PPT algorithms $\mathcal{A}$ given g, g^a, g^b, g^u, g^v and g^{au+bv} it is difficult to calculate g^{au}, i.e. it holds that*

$$\Pr\left[\mathcal{A}(g, g^a, g^b, g^u, g^v, g^{au+bv}) = g^{au} \,:\, a, b, u, v \leftarrow \mathbb{Z}_p\right] \leq \mathsf{negl}(\kappa)$$

for a function negl*, which is negligible in the security parameter κ.*

The connection to the BGLS aggregate signature scheme is as follows (see Section 3.2.1 for an overview of the scheme):

- g^a and g^b can be interpreted as public keys of two different parties using the BGLS aggregate signature scheme.
- g^u and g^v can be interpreted as the hash values of two distinct messages m_1 and m_2, i.e. $H(m_1) = g^u$ and $H(m_2) = g^v$.
- This implies that $H(m_1)^a = g^{au}$ and $H(m_2) = g^{bv}$ are valid signatures for m_1 and m_2, therefore g^{au+bv} is an aggregate signature of m_1 and m_2 under the public keys of the two parties.
- The goal of the attacker is to extract g^{au} from g^{au+bv}, i.e. the attacker needs to deaggregate one of the individual signatures from the given aggregate signature, without knowledge of any individual signature.

Boneh et al. only state this assumption, but give no explicit justification to assume that this problem is hard. Fortunately, in [CN03] Coron and Naccache prove that the k-Element Aggregate Extraction assumption is equivalent to the well-known Computational Diffie-Hellman assumption (for all $k \geq 2$).

Theorem 5.2.2. *The* 2EAE *assumption holds in $\mathbb{G}$ if and only if the Computational Diffie-Hellman Assumption also holds in $\mathbb{G}$.*

Proof. A proof of this theorem can be found in [CN03]. □

This assumption is also sufficient for Boneh et al. [Bon+03] to be able to construct a verifiably encrypted signature scheme from their aggregate signature scheme. This motivates our basic form of deaggregation security defined in the following section.

5.2.2 nDEAGG Security

We first present a definition that is strongly motivated by the 2EAE assumption of Boneh et al. [Bon+03]. The attacker can send n messages to the challenger, for which he computes n individual signatures, aggregates them and hands the resulting aggregate signature over to the attacker. The goal of the attacker is then to output an aggregate signature for a subset of these messages. As it turns out, for many "natural" schemes, security for the special case of two messages also implies security for any number of messages (see Section 5.2.5). We formulate individual security definitions for each n instead of a more general definition, since schemes might only be deaggregation secure for specific numbers of claims, which is also further discussed in Section 5.2.5.

Remark. For the main part of this chapter, we focus on the case of fully flexible aggregate signatures. We discuss how to adapt the definitions to sequential and synchronized aggregate signature schemes in Section 5.2.8.

Unlike in the definitions for unforgeability, we need to strongly restrict the knowledge of the attacker concerning the secret keys, i.e. he is not allowed to choose any of the secret keys. This rules out trivial attacks, since if even one secret key would be known to the attacker, he could trivially "deaggregate" by simply signing a message under this key and outputting this newly computed signature. However, in Section 5.2.3 we also present a definition in which attackers are allowed to use keys and claims of their choosing *additionally* to the keys given by the challenger.

For this definition, we have to take into account that the order in which the claims are aggregated might influence the resulting aggregate signature. Aggregating the same set of claims in different orders could potentially produce different signatures, which for example is the case in the scheme of Lysyanskaya et al. [Lys+04]. Here, the order of aggregation decides the order in which several trapdoor permutations are evaluated on the messages. Therefore, the resulting aggregate signature is strongly influenced by the order of aggregation[1] (see Section 5.3.4 for details of the scheme). Theoretically, it could therefore be possible that certain orders of aggregation result in aggregate signatures which are "easier to deaggregate" than others. To reflect this, we allow the attacker to specify the order in which the signatures are to be aggregated by sending an *order-tree* as defined next:

Definition 5.2.3 (Order-Tree)**.** *An* order-tree T *for a claim sequence* C *with* $|C| = n$*, i.e. containing* n *claims, is a binary tree with* n *leaves, where the edges and leaves are marked as follows:*

- *The two outgoing edges of each inner node are marked with* 1 *and* 2*.*
- *The leaves are marked with pairwise distinct numbers in* $\{1, \ldots, n\}$*.*

An order-tree T implies a fully specified order of aggregation by applying the following algorithm OrderAgg to the tree T, the corresponding claim sequence C and list of secret keys.

[1] Recall that permutations in general do not commutate.

Algorithm 1. $\mathsf{OrderAgg}(\mathsf{sk}, C, T)$

input : A list of secret keys $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$, a claim sequence $C = (C_1, \ldots, C_n)$ with $C_i = (\mathsf{pk}_i, m_i)$ and an order-tree T
output : An aggregate signature σ_{Agg} valid for C

for $i = 1, \ldots, n$ **do**
  $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m_i)$
  $\mathcal{C}_i := (C_i)$
$k := n + 1$
while T *has more than one node* **do**
  Choose a pair of leaves a and b with a common successor c.
  W.l.o.g. let a be connected to c by the edge marked with 1.
  $i :=$ marking of a
  $j :=$ marking of b
  $\sigma_k \leftarrow \mathsf{Agg}(C_i, C_j, \sigma_i, \sigma_j)$
  Delete a and b from T
  Mark node c with k
  $\mathcal{C}_k = \mathcal{C}_i \| \mathcal{C}_j$
  $k := k + 1$
return $\sigma_{\mathsf{Agg}} := \sigma_{k-1}$

Example 5.2.4. Let $C = (C_1, C_2, C_3) = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2), (\mathsf{pk}_3, m_3))$. Let T be the order-tree depicted in Figure 5.1.

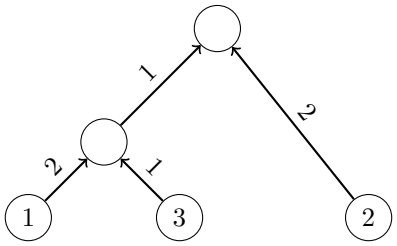

Figure 5.1: An example order-tree for three claims.

Then, according to T, the algorithm $\mathsf{OrderAgg}$ would sign and aggregate the claims in the following order:

1. Create individual signatures $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m_i)$ for $i \in \{1, 2, 3\}$.

2. Aggregate σ_1 and σ_3 by computing $\sigma_4 \leftarrow \mathsf{Agg}((c_3), (c_1), \sigma_3, \sigma_1)$. Observe that the order of the claims and signatures in the input of Agg follows the order depicted in the tree.

3. Aggregate σ_4 and σ_2 by computing $\sigma_5 \leftarrow \mathsf{Agg}((c_3, c_1), (c_2), \sigma_4, \sigma_2)$.

4. Output the resulting aggregate signature $\sigma_{\mathsf{Agg}} := \sigma_5$.

Using order-trees, we can easily define the $n\mathsf{DEAGG}$ security experiment. A schematic overview of the experiment can be found in Figure 5.2. Note that we fix the number n of messages and thereby build a hierarchy of security definitions instead of giving the attacker the option to choose n. We define security in this way, since the relationship between the levels of different n is surprisingly complex, as we discuss in Section 5.2.5.

Definition 5.2.5 ($n\mathsf{DEAGG}$ Security Experiment)**.** *The $n\mathsf{DEAGG}$ security experiment for $n \in \mathbb{N}, n \geq 2$, between an attacker $\mathcal{A}$, a challenger $\mathcal{C}$ and an aggregate signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ consists of three phases as follows:*

Setup Phase. *The challenger $\mathcal{C}$ generates n key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\kappa)$, gives the public keys pk_i to the attacker $\mathcal{A}$ and stores the secret keys $\mathsf{sk} := (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.*

Challenge Phase. *The attacker $\mathcal{A}$ sends a list of exactly n messages $\mathcal{M} = (m_1, \ldots, m_n)$ and an order-tree T to the challenger. The challenger then sets $C := ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$, computes*

$$\sigma_{\mathsf{Agg}} \leftarrow \mathsf{OrderAgg}(\mathsf{sk}, C, T)$$

and sends σ_{Agg} to $\mathcal{A}$. This step may only be executed once *by the attacker.*

Deaggregation Phase. *At the end of the experiment, $\mathcal{A}$ sends a tuple (C^*, σ^*) consisting of a claim sequence C^* and a signature σ^* to the challenger $\mathcal{C}$. $\mathcal{A}$ is* successful*, if*

$$\mathsf{Vfy}(C^*, \sigma^*) = 1 \text{ and } 0 < |C^*| \leq n - 1 \text{ and } \forall c_i \in C^* : c_i \in C.$$

The last two parts of the success requirement enforce that

1. $\mathcal{A}$ must at least deaggregate one signature from σ_{Agg}.
2. $\mathcal{A}$ may not add new claims to win the experiment. The reason for this restriction is that there are schemes which are $n\mathsf{DEAGG}$ secure according to this definition, but not secure if we allow the attacker to add such additional claims (see Section 5.3 for a discussion of several of such schemes, like the scheme of [Bon+03] discussed in Section 5.3.1). It is therefore advantageous to be able distinguish these two cases. In the following Section 5.2.3 we introduce a variation of this definition without this restriction.

Definition 5.2.6 ($n\mathsf{DEAGG}$ Security for Aggregate Signature Schemes)**.** *An aggregate signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{AggVfy})$ is $n\mathsf{DEAGG}$ secure, if all PPT algorithms $\mathcal{A}$ only have negligible success probability in the $n\mathsf{DEAGG}$ security experiment, meaning it holds that*

$$\Pr\left[\mathcal{A}^{\mathcal{C}_{n\mathsf{DEAGG}}}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) = (C^*, \sigma^*) : \begin{array}{c} \mathsf{Vfy}(C^*, \sigma^*) = 1 \\ \wedge\, 0 < |C^*| \leq n - 1 \\ \wedge\, \forall c_i \in C^* : c_i \in C \end{array}\right] \leq \mathsf{negl}(\kappa)$$

for a function negl, which is negligible in the security parameter κ.

Remark. Note that in the definition of $n\mathsf{DEAGG}$ security, the attacker *does not* have access to a general signing oracle. The only way for the attacker to receive signatures is by issuing the single challenge query. In Section 5.2.6 we discuss the reasons and motivations for this restriction and in Section 5.2.7 a stronger notion is presented that allows adaptive queries.

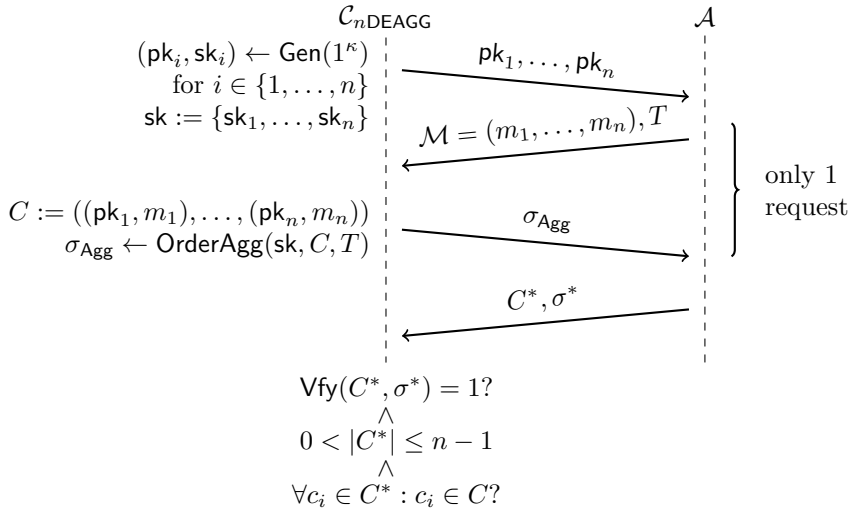

Figure 5.2: The $n\mathsf{DEAGG}$ security experiment.

5.2.3 $n\mathsf{DEAGG}^+$ Security: Attacks using Additional Claims

The definition of $n\mathsf{DEAGG}$ security rules out attackers that might try to deaggregate by *aggregating* additional claims of their own choosing to the challenge signature, since the definitions requires the claim sequence C^* returned by the attacker to be a subset of the original claim sequence. In particular, this implies that the attacker may not use any freely chosen keys for the attack. For many applications, requiring that no additional keys are used seems like a reasonable restriction, since the set of admissible public keys might be fixed. For example, in a sensor network, the public keys of all sensors should be known to its operators. For more general applications, it might be required that all keys need to be registered in a public key infrastructure. Then, being able to manipulate an aggregate signature so that it is also valid for a *new* claim under such a registered key is equivalent to breaking the unforgeability of the signature scheme, as long as the attacker has no knowledge of its secret key.

But even in these scenarios where the set of keys is fixed and controlled, the attacker could still try to use other public keys to deaggregate. Although this would be noticeable and covert attacks using this approach are not possible, they might still pose a serious risk. While it would be clear which claims were added maliciously, it might not be obvious if other claims where deaggregated during the attack and if so, which ones were deaggregated.

For example, suppose there exists a signature σ for a claim sequence $C = (c_1, c_2)$ with $c_1 = (\mathsf{pk}_1, m_1)$, $c_2 = (\mathsf{pk}_2, m_2)$ and pk_1, pk_2 are the only admissible public keys. If the attacker can manipulate σ so that it becomes valid for the claim sequence $((\mathsf{pk}_1, m_1), (\mathsf{pk}_3, m_3))$ for a new public key pk_3 and (possibly) new message m_3, the system would easily recognize that pk_3 is not among the admissible keys. However, it is no longer clear if the signature should be interpreted as a valid signature of m_1 and if m_2 was ever signed or not. Also,

if the attacker knows the secret key to pk_3, he might even be able to legally register it before he mounts his attack.

At first glance, it might also seem paradoxical that aggregating new claims could help the attacker to deaggregate. But, for example, the signatures of many schemes are elements of algebraic groups and they are aggregated by multiplying them in the group. Here, aggregating a new claim can clearly influence the already aggregated ones and, in fact, many schemes are vulnerable to such attacks, as we show in Section 5.3.

We therefore extend our definition of $n\mathsf{DEAGG}$ security to $n\mathsf{DEAGG}^+$ security, which allows the attacker to use *additional* claims and public keys of his choosing. However, so that the notion captures a reasonable definition of deaggregation security, at least one of the original claims must be present in the claim sequence output by the attacker.

Another justification for this additional notion is that several schemes can be shown to be $n\mathsf{DEAGG}$ secure, but are vulnerable to attacks using additional claims and keys, as is also shown in Section 5.3. The notion of $n\mathsf{DEAGG}^+$ security therefore seems to be strictly stronger then $n\mathsf{DEAGG}$ security. The security experiment for $n\mathsf{DEAGG}^+$ security is almost identical to the $n\mathsf{DEAGG}$ security experiment, only the winning condition is changed:

Definition 5.2.7 ($n\mathsf{DEAGG}^+$ Security Experiment)**.** *The* $n\mathsf{DEAGG}^+$ security experiment *for* $n \in \mathbb{N}, n \geq 2$ *between an attacker* $\mathcal{A}$*, a challenger* $\mathcal{C}$ *and an aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *consists of three phases as follows:*

Setup Phase. *The challenger* $\mathcal{C}$ *generates* n *key pairs* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\kappa)$*, gives the public keys* pk_i *to the attacker* $\mathcal{A}$ *and stores the secret keys* $\mathsf{sk} := (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$*.*

Challenge Phase. *The attacker* $\mathcal{A}$ *sends a list of exactly* n *messages* $\mathcal{M} = (m_1, \ldots, m_n)$ *and an order-tree* T *to the challenger. The challenger then sets* $C := ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$*, computes*

$$\sigma_{\mathsf{Agg}} \leftarrow \mathsf{OrderAgg}(\mathsf{sk}, C, T)$$

and sends σ_{Agg} *to* $\mathcal{A}$*. This step may only be executed* once *by the attacker.*

Deaggregation Phase. *At the end of the experiment,* $\mathcal{A}$ *sends a tuple* (C^*, σ^*) *consisting of a claim sequence* C^* *and a signature* σ^* *to the challenger* $\mathcal{C}$*.* $\mathcal{A}$ *is* successful*, if*

$$\mathsf{Vfy}(C^*, \sigma^*) = 1 \;\; \textit{and} \; \exists c \in C : c \notin C^* \; \textit{and} \; \exists c \in C^* : c \in C.$$

Definition 5.2.8 ($n\mathsf{DEAGG}^+$ Security for Aggregate Signature Schemes)**.** *An aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{AggVfy})$ *is* $n\mathsf{DEAGG}^+$ secure*, if all PPT algorithms* $\mathcal{A}$ *only have negligible success probability in the* $n\mathsf{DEAGG}^+$ *security experiment, meaning it holds that*

$$\Pr\left[\mathcal{A}^{\mathcal{C}_{n\mathsf{DEAGG}^+}}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) = (C^*, \sigma^*) \; : \; \begin{array}{c} \mathsf{Vfy}(C^*, \sigma^*) = 1 \\ \wedge \exists c_i \in C \;\; : c \notin C^* \\ \wedge \exists c_i \in C^* : c \in C \end{array}\right] \leq \mathsf{negl}(\kappa)$$

for a function negl*, which is negligible in the security parameter* κ*.*

Note the differences in the success requirement compared to the definition of $n\mathsf{DEAGG}$ security (see definitions 5.2.5 and 5.2.6). Instead of requiring that each claim in C^* has to be equal to one of the claims of the original claim sequence C, we now only require that there exists at least one such claim. This allows the attacker to add additional claims, as desired, while ensuring that the deaggregation is not trivial (i.e. the attacker simply outputs a completely new claim sequence). We further require that there is at least one claim in C which is *not* in C^* to enforce that at least one claim was actually deaggregated. Observe also that C^* may now contain more than $n-1$ claims.

Since $n\mathsf{DEAGG}^+$ security is a straightforward extension of $n\mathsf{DEAGG}$ security and a successful $n\mathsf{DEAGG}$ attack also breaks $n\mathsf{DEAGG}^+$ security, we state the following theorem without formally proving it:

Theorem 5.2.9. *If an aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *is* $n\mathsf{DEAGG}^+$ *secure for* $n \in \mathbb{N}$*, then it is also* $n\mathsf{DEAGG}$ *secure.*

5.2.4 Deaggregation Security & Unforgeability

A natural question is to ask whether there is a relationship between the unforgeability of a given aggregate signature scheme and its deaggregation security, especially if the scheme also achieves optimal compression (i.e. aggregate signatures do not grow in size and have the same size as individual signatures). First, note that neither $n\mathsf{DEAGG}$ nor $n\mathsf{DEAGG}^+$ security imply any of the general unforgeability definitions presented in this thesis, since these definitions allow only one single Sign query.

Second, the answer to the question if unforgeability implies $n\mathsf{DEAGG}$ or $n\mathsf{DEAGG}^+$ security is unfortunately also negative, as we show in this section. This is especially surprising for the case of sequential aggregate signatures that are secure in the sense of the $\mathsf{SAS\text{-}EUF\text{-}CMA}$ unforgeability definition of Lysyanskaya et al. [Lys$^+$04]. Here, the attacker does not necessarily need to output a signature for a new message. If he asks for a signature on $C\|(\mathsf{pk}^*, m^*)$ from the challenger and then is able to modify it, so that it is valid for another "prefix" $C' \neq C$ (i.e. he can compute a signature valid for $C'\|(\mathsf{pk}^*, m^*)$), then he would also win the experiment, although he would reuse the message m^*. One such manipulation of the prefix C could of course be the deaggregation of a claim from C. Even though this seems to imply that $\mathsf{SAS\text{-}EUF\text{-}CMA}$ security already encompasses some restricted form of deaggregation security, we can show that it does not imply $n\mathsf{DEAGG}$ security. See Definition 3.3.4 and the discussion below it for more details on $\mathsf{SAS\text{-}EUF\text{-}CMA}$ security.

We now first show that unforgeability does not imply deaggregation security for fully flexible aggregate signature schemes. For this purpose, let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ be an $\mathsf{AS\text{-}EUF\text{-}CMA}$ secure aggregate signature scheme with optimal compression. We use it to construct a new aggregate signature scheme $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Agg}', \mathsf{Vfy}')$, which also achieves optimal compression and is $\mathsf{AS\text{-}EUF\text{-}CMA}$ secure, but is not $n\mathsf{DEAGG}$ secure for any n.

The main idea is that signatures of Σ' simply are tuples $\sigma_{\mathsf{Agg}} = (\sigma_1, \sigma_2)$, where σ_1 and σ_2 are signatures of the scheme Σ. The entry σ_1 will be an aggregate signature over the whole claim sequence, whereas σ_2 is an aggregate signature of one of the two claim sequences that were aggregated to create σ_{Agg}.

This scheme also achieves optimal compression, since none of the elements grow in size during aggregation. However, it is not deaggregation secure, since the second entry σ_2 of σ_{Agg} can be used to construct a valid signature for a subsequence. We explain this in more detail in Lemma 5.2.11, but first, we formally define the algorithms of Σ' and prove its unforgeability:

$\mathsf{Gen}'(1^\kappa)$: Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ and return $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}'(\mathsf{sk}, m)$: Compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ and return (σ, σ).

$\mathsf{Agg}'(C_1, C_2, \sigma_1, \sigma_2)$: Parse $\sigma_i = (\sigma_{i,1}, \sigma_{i,2})$. Return

$$\sigma_{\mathsf{Agg}} = (\mathsf{Agg}(C_1, C_2, \sigma_{1,1}, \sigma_{2,1}), \sigma_{1,1}).^2$$

$\mathsf{Vfy}'(C, \sigma)$: Parse $\sigma = (\sigma_1, \sigma_2)$. Ignore σ_2 and output $\mathsf{Vfy}(C, \sigma_1)$.

If Σ achieves optimal compression, then so does Σ', since signatures of Σ' are just tuples of Σ and no other elements are added. So Σ' signatures are twice the size of Σ, but their size never grows. The correctness of Σ' follows directly from the correctness of Σ.

Observe that the structure of the algorithms, if they are applied correctly, ensures that there never exist multiple individual signatures for different messages of Σ under the same key. While this is not important for most fully flexible schemes, it is crucial for synchronized schemes, were signers are only allowed to sign once per time period. For example, the synchronized scheme of Ahn, Green, and Hohenberger [AGH10] (see Section 5.3.7 for details) becomes insecure if more than one signature under the same public key exists per time period.

Lemma 5.2.10. *If Σ is* AS-EUF-CMA *secure, then so is Σ'.*

Proof. The proof of this theorem is a straightforward reduction, see Figure 5.3 for an overview. Let $\mathcal{C}_{\mathsf{AS\text{-}EUF\text{-}CMA}}$ be the AS-EUF-CMA challenger for Σ and $\mathcal{A}$ a PPT attacker on the AS-EUF-CMA security of Σ'. We construct a simulator $\mathcal{B}$ which uses $\mathcal{A}$ to break Σ.

$\mathcal{C}_{\mathsf{AS\text{-}EUF\text{-}CMA}}$ creates a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$ and sends pk to $\mathcal{B}$, who in turn forwards pk to $\mathcal{A}$. If $\mathcal{A}$ sends a query for a message m_i, $\mathcal{B}$ sends m_i to the challenger, receives a Σ signature σ for m_i and sends $\sigma' := (\sigma, \sigma)$ to $\mathcal{A}$. Since σ is a valid Σ signature for m_i, it follows that σ' is a valid Σ' signature for m_i.

At some point, $\mathcal{A}$ sends its forgery (C^*, σ^*) with $\sigma^* = (\sigma_1^*, \sigma_2^*)$. If $\mathcal{A}$ is successful, then we have $\mathsf{Vfy}'(C^*, \sigma^*) = 1$, which implies $\mathsf{Vfy}(C^*, \sigma_1^*) = 1$. Furthermore, there exists a claim $(\mathsf{pk}, m^*) \in C^*$, such that m^* was never sent in a signature query by $\mathcal{A}$ and therefore $\mathcal{B}$ also never sent it to the challenger $\mathcal{C}_{\mathsf{AS\text{-}EUF\text{-}CMA}}$. $\mathcal{B}$ then outputs C^*, σ_1^* to win the AS-EUF-CMA experiment.

$\mathcal{B}$ simulates the AS-EUF-CMA experiment perfectly for $\mathcal{A}$ and the success probability of $\mathcal{B}$ is at least the same as the success probability of $\mathcal{A}$. Their runtimes are roughly the same ($\mathcal{B}$ only has a small polynomial overhead compared to $\mathcal{A}$), therefore $\mathcal{B}$ is also a PPT algorithm. It follows that the success probability of $\mathcal{A}$ must be negligible, since we assumed Σ to be AS-EUF-CMA secure, which proves the theorem. □

[2] Observe that for honest computations, the second entry of an aggregate signature now always contains a valid aggregate signature for the claim sequence C_1 used to compute the signature.

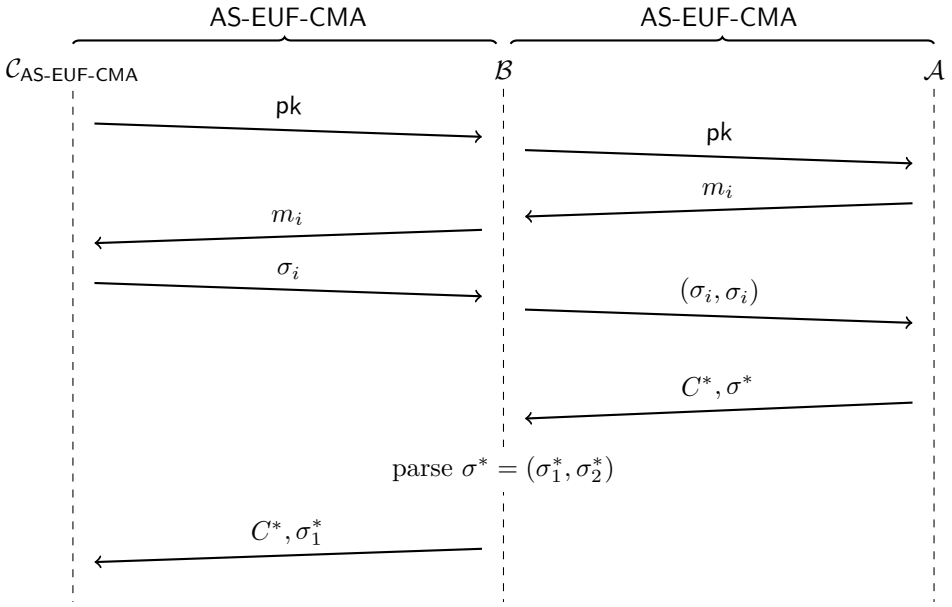

Figure 5.3: Overview of the proof strategy for Lemma 5.2.10.

Lemma 5.2.11. Σ' *is* not nDEAGG *secure for any* n*.*

Proof. Consider the following PPT attacker $\mathcal{A}$ on the nDEAGG experiment. First, $\mathcal{A}$ receives a list of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ from the challenger $\mathcal{C}_{n\mathsf{DEAGG}}$. $\mathcal{A}$ then chooses n messages $m_1, \ldots, m_n$ at random and sends $(m_1, \ldots, m_n), T$ to $\mathcal{C}_{n\mathsf{DEAGG}}$, where T is some fitting order-tree for the n messages (for example the order-tree which describes a sequential aggregation). Next, $\mathcal{A}$ receives a valid Σ' aggregate signature $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2})$. Since σ_{Agg} is valid, we have that $\mathsf{Vfy}(C, \sigma_{\mathsf{Agg},1}) = 1$. Furthermore, since σ_{Agg} was *honestly* computed, we also have that $\mathsf{Vfy}(C', \sigma_{\mathsf{Agg},2}) = 1$, where C' is a claim sequence containing only (but *not* all) claims of C, i.e. $|C'| < |C| = n$.

To be precise, C' is one of the two claim sequences used in the last aggregation step of the challenger to compute σ_{Agg}. Since $\mathcal{A}$ chose the order-tree T himself, he also knows these two claim sequences and therefore also knows C', even though it might not be possible to deduce it from σ_{Agg}.[3] $\mathcal{A}$ now sends $C', \sigma^* \coloneqq (\sigma_{\mathsf{Agg},2}, \bot)$ as its answer to the challenger

Note that since all claims of C' are also in C and we have $|C'| < |C| = n$, this is a valid deaggregation. Furthermore, while σ^* is not a regular signature for C', it is still valid, since the Vfy' algorithm simply ignores the second entry of the tuple and we therefore have $\mathsf{Vfy}'(C', \sigma^*) = 1$, since $\mathsf{Vfy}(C', \sigma_{\mathsf{Agg},2}) = 1$. All in all, we see that $\mathcal{A}$ wins the experiment.

Furthermore, the runtime of $\mathcal{A}$ is polynomial and its success probability is equal to 1, which proves that the scheme is not nDEAGG secure. □

[3] While in some schemes it *is* possible to efficiently extract the signed claims from the signature (for example [Lam79; Nev08]), this is not the case in general.

Theorem 5.2.12. AS-EUF-CMA *security does* not *imply* nDEAGG *or* nDEAGG$^+$ *security.*

Proof. The theorem follows directly from Theorem 5.2.9 and the Lemmas 5.2.10 and 5.2.11 above. □

While one could argue that this counter-example is pathological, since the scheme was clearly designed to be vulnerable to deaggregation attacks, it unmistakably proves that nDEAGG security does not follow directly from AS-EUF-CMA security. Similar counter-examples can be given for sequential and synchronized aggregate signatures as well. In fact, for synchronized schemes, no change in the counter-example is necessary.

For sequential aggregation, the counter-example needs to be minimally modified, because of the different syntax of the algorithms. To be precise, let $\Sigma_{\mathsf{SAS}} = (\mathsf{Gen}_{\mathsf{SAS}}, \mathsf{AggSign}_{\mathsf{SAS}}, \mathsf{Vfy}_{\mathsf{SAS}})$ be a sequential aggregate signature scheme with optimal compression. Then the algorithms for the counter-example $\Sigma'_{\mathsf{SAS}} = (\mathsf{Gen}'_{\mathsf{SAS}}, \mathsf{AggSign}'_{\mathsf{SAS}}, \mathsf{Vfy}'_{\mathsf{SAS}})$ are as follows:

$\mathsf{Gen}'_{\mathsf{SAS}}(1^\kappa)$: Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}_{\mathsf{SAS}}(1^\kappa)$ and return $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{AggSign}'_{\mathsf{SAS}}(\mathsf{sk}, C, \sigma, m)$: Parse $\sigma = (\sigma_1, \sigma_2)$. Compute

- $\sigma_{\mathsf{Agg},1} \leftarrow \mathsf{AggSign}_{\mathsf{SAS}}(\mathsf{sk}, C, \sigma_1, m)$,
- $\sigma_{\mathsf{Agg},2} \coloneqq \sigma_1$,

and output $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2})$.

$\mathsf{Vfy}'_{\mathsf{SAS}}(C, \sigma)$: Parse $\sigma = (\sigma_1, \sigma_2)$. Ignore σ_2 and return $\mathsf{Vfy}_{\mathsf{SAS}}(C, \sigma_1)$.

Σ' has optimal compression and is SAS-EUF-CMA secure, but is not nDEAGG secure for any n. The proofs are largely analogous to the proofs of the fully flexible counter-example.

Lemma 5.2.13. *If* Σ_{SAS} *is* SAS-EUF-CMA *secure, then so is* Σ'_{SAS}.

Proof. Let $\mathcal{C}_{\mathsf{SAS\text{-}EUF\text{-}CMA}}$ be the SAS-EUF-CMA challenger for Σ_{SAS} and $\mathcal{A}$ a PPT attacker on the SAS-EUF-CMA security of Σ'. We construct a simulator $\mathcal{B}$ which uses $\mathcal{A}$ to break Σ_{SAS}.

$\mathcal{C}_{\mathsf{SAS\text{-}EUF\text{-}CMA}}$ creates a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$ and sends pk to $\mathcal{B}$, who in turn forwards pk to $\mathcal{A}$. If $\mathcal{A}$ sends a signature query for C_i, σ_i, m_i, $\mathcal{B}$ forwards it to the challenger. $\mathcal{B}$ then receives a Σ_{SAS} signature σ for $C_i \| (\mathsf{pk}, m_i)$ and sends $\sigma' \coloneqq (\sigma, \sigma_i)$ to $\mathcal{A}$. Since σ is a valid Σ_{SAS} signature for m_i, it follows that σ' is a valid Σ'_{SAS} signature for m_i.

At some point, $\mathcal{A}$ sends its forgery (C^*, σ^*) with $\sigma^* = (\sigma_1^*, \sigma_2^*)$. If $\mathcal{A}$ is successful, then $\mathsf{Vfy}'_{\mathsf{SAS}}(C^*, \sigma^*) = 1$, which implies that $\mathsf{Vfy}_{\mathsf{SAS}}(C^*, \sigma_1^*) = 1$, and C^* is non-trivial. Since the signature queries of $\mathcal{B}$ are equal to the queries of $\mathcal{A}$, it is also non-trivial for $\mathcal{B}$. Therefore, $\mathcal{B}$ simply outputs C^*, σ_1^* to win the experiment.

$\mathcal{B}$ simulates the SAS-EUF-CMA experiment perfectly for $\mathcal{A}$ and the success probability of $\mathcal{B}$ is at least the same as the success probability of $\mathcal{A}$. Their runtimes are roughly the same ($\mathcal{B}$ only has a small polynomial overhead compared to $\mathcal{A}$), therefore $\mathcal{B}$ is also a PPT algorithm. It follows that the success probability of $\mathcal{A}$ must be negligible, since we assumed Σ_{SAS} to be SAS-EUF-CMA secure, which proves the theorem. □

Lemma 5.2.14. Σ'_{SAS} *is* not nDEAGG *secure for all* $n \geq 2$ *polynomial in the security parameter* κ.

Proof. Consider the following PPT attacker $\mathcal{A}$ on the nDEAGG experiment. First, $\mathcal{A}$ receives a list of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ from the challenger $\mathcal{C}_{n\mathsf{DEAGG}}$. $\mathcal{A}$ then chooses n messages $m_1, \ldots, m_n$ at random and sends $(m_1, \ldots, m_n), T$ to $\mathcal{C}_{n\mathsf{DEAGG}}$, where T describes the sequential aggregation in ascending order of m_1 to m_n. Next, $\mathcal{A}$ receives a valid Σ'_{SAS} aggregate signature $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2})$.

Let $C \coloneqq (c_1, \ldots, c_n) \coloneqq ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$. Since σ_{Agg} is valid, we have that $\mathsf{Vfy}_{\mathsf{SAS}}(C, \sigma_{\mathsf{Agg},1}) = 1$. Furthermore, since σ_{Agg} was *honestly* computed, we also have $\mathsf{Vfy}_{\mathsf{SAS}}(C \setminus (c_n), \sigma_{\mathsf{Agg},2}) = 1$.

$\mathcal{A}$ now sends $C^* \coloneqq C \setminus (c_n)$ and $\sigma^*_{\mathsf{Agg}} \coloneqq (\sigma_{\mathsf{Agg},2}, \bot)$ as its answer to the challenger.

$\mathsf{Vfy}'_{\mathsf{SAS}}(C^*, \sigma^*) = 1$ follows from $\mathsf{Vfy}_{\mathsf{SAS}}(C^*, \sigma_{\mathsf{Agg},2}) = 1$ and since c_n was removed from C, but all other claims of C are still present in C^*, it is also a valid deaggregation. The runtime of $\mathcal{A}$ is polynomial and its success probability is equal to 1.

Note that while σ^* is not a regular signature for the claim sequence C^*, it is still valid, since $\mathsf{Vfy}'_{\mathsf{SAS}}$ does not verify the second entry of the signature tuple. □

All in all, the following theorem follows:

Theorem 5.2.15. SAS-EUF-CMA *security and* SyncAS-EUF-CMA *do* not *imply* nDEAGG *or* nDEAGG$^+$ *security.*

Additionally, for sequential aggregate signatures, "real-life counter-examples", albeit only in the random oracle model, are also given by the schemes of Lysyanskaya et al. [Lys+04] and Neven [Nev08], which we discuss in Sections 5.3.4 and 5.3.5.

5.2.5 Relationships between nDEAGG and nDEAGG$^+$

In this section we discuss the relationships between different levels of nDEAGG and nDEAGG$^+$ security. As it turns out, for certain aggregate signature schemes that fulfill three rather natural requirements, which we call *extendability*, *claim-removability* and *order-independence*, we can show that 2DEAGG security implies nDEAGG security for all n polynomial in the security parameter κ. So, for these schemes analyzing the simplest form of nDEAGG security suffices. The proof strategy can also be applied to show that nDEAGG security implies n'DEAGG security for $n' > n$ for these schemes. Moreover, we also show that in general nDEAGG security does not imply n'DEAGG security for $n' \neq n$ if these requirements are not met. All of these implications can also be shown for nDEAGG$^+$ security.

Remark. All definitions, theorems and lemmas can be adapted to sequential and synchronized aggregate schemes. We focus on fully flexible aggregate signatures for sake of brevity. Section 5.2.8 explains the necessary adaptations. However, some theorems do not apply to sequential schemes which are SAS-EUF-CMA secure as defined by [Lys+04], since these schemes cannot be claim-removable and order-independent (see below for details).

We first discuss in which cases 2DEAGG implies nDEAGG security. The requirements for this implication to hold are:

Extendability. It must always be possible to further aggregate claims into an already aggregated signature. This requirement is fulfilled by almost all aggregate signature schemes and stands in direct connection to the ability to aggregate. However, some schemes might restrict aggregation in some way, for example by imposing an upper bound on the number of signatures which can be aggregated (like our fault-tolerant scheme from Chapter 4) and are therefore not *extendable*.

Claim-Removability. It must be possible for the owner of the secret key to efficiently deaggregate their signature from an aggregate.

This might seem counter-intuitive, since we want to proof security against deaggregation attacks. The crucial difference here is the knowledge of the secret key. Deaggregation should only be possible if the secret key is known and hard otherwise. Several schemes like [Bon+03; Lu+06; AGH10] fulfill this property. [Lu+06] actually mention it as a feature: In their scheme, each signer may only aggregate one individual signature into an aggregated signature. They explicitly point out that this is not a substantial restriction, since if a signer wants to add a signature for a second message, they can deaggregate their current signature and then aggregate a new signature for both the first and second message.

Order-Indepence. The order in which signatures are aggregated is not consequential and cannot be inferred from an aggregate signature.

Note that these properties might also be of independent interest. For example, some applications might require claim-removability to be able to efficiently make use of an aggregate signature scheme. Consider the case of software validation on a mobile device, where only signed apps are allowed to run. If an aggregate signature scheme is used to sign and verify apps, there must be some way to securely remove the signature of an app if the user wants to de-install it. On the other hand, other applications might require that deaggregation even is hard for the signers, for example if the scheme is used to sign contracts. If signers could simply remove their signatures, this would imply that they can void contracts at any point in time.

All three properties seem to be essential for the proof of Theorem 5.2.21 that shows that 2DEAGG implies nDEAGG security for schemes that fulfill all three properties. As we show at the end of this section, Theorem 5.2.27 implies that order-independence is a necessary requirement. The same is not known for extendability and claim-removability. Although we cannot rule out that it might be possible to show the implication without these properties, we are also not aware of any such proofs and this problem remains open for now.

We now formally define the described properties in regard to aggregate signature schemes. For the following definitions, let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ be an aggregate signature scheme.

Definition 5.2.16 (Extendability)**.** *Let σ_{Agg} be an aggregate signature for a claim sequence $C = (c_1, \ldots, c_n)$ and c a claim with a corresponding individual signature σ. Σ is* extendable*, if given $\sigma_{\mathsf{Agg}}, C, c$ and σ it is possible to create an aggregate signature σ'_{Agg} for the claim sequence $C' = C \| (c)$ in polynomial time in the security parameter κ.*

For schemes that only allow distinct public keys during aggregation (like the scheme of [Lu+06], see Section 5.3.6), the definition is slightly changed, so that we only require that claims c that contain a public key which is not yet present in C can be aggregated.

Definition 5.2.17 (Claim-Removability)**.** *Let σ_{Agg} be an aggregate signature for a claim sequence $C = (c_1, \ldots, c_n)$ and let sk_i be the secret key corresponding to claim c_i. Σ is* claim-removable*, if given σ_{Agg}, C and sk_i it is possible to compute a* valid *aggregate signature σ'_{Agg} for the claim sequence $C' = C \setminus (c_i)$ in polynomial time in the security parameter κ.*

We define order-independence by using a fitting security experiment.

Definition 5.2.18 (Order-Independence Experiment)**.** *The* order-independence experiment *between an attacker $\mathcal{A}$, a challenger $\mathcal{C}$ and Σ consists of three phases as follows:*

Setup Phase. *$\mathcal{A}$ sends a number $n \in \mathbb{N}$ to $\mathcal{C}$. $\mathcal{C}$ then generates n key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\kappa)$ and gives all public keys pk_i to the attacker $\mathcal{A}$ and stores all secret keys $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.*

Challenge Phase. *The attacker $\mathcal{A}$ sends a claim sequence of exactly n claims $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$ and two order-trees T_0, T_1 to the challenger. The challenger then chooses $b \leftarrow \{0, 1\}$ randomly, computes $\sigma_{\mathsf{Agg}} \leftarrow \mathsf{OrderAgg}(\mathsf{sk}, C, T_b)$ and sends σ_{Agg} to $\mathcal{A}$. This step may only be executed* once *by the attacker.*

Answer Phase. *At the end of the experiment, $\mathcal{A}$ sends a bit b'. $\mathcal{A}$ is* successful*, if $b = b'$.*

Definition 5.2.19 (Order-Independence)**.** *Σ is* order-independent*, if for all PPT algorithms $\mathcal{A}$ it holds that*

$$\begin{aligned} |\Pr[\mathcal{A}^{\mathcal{C}_{\mathsf{OrdInd}}}((\mathsf{pk}_1, \ldots \mathsf{pk}_n) = 0 \,:\, b = 0] \\ - \Pr[\mathcal{A}^{\mathcal{C}_{\mathsf{OrdInd}}}((\mathsf{pk}_1, \ldots, \mathsf{pk}_n) = 0 \,:\, b = 1]| \leq \mathsf{negl}(\kappa) \end{aligned}$$

for a function negl, which is negligible in the security parameter κ.

Note that order-independence does not imply that it is completely impossible for an attacker to infer some information of the order of the aggregation. It only implies that the structure of an aggregate signature does not reveal any information of the order in which the claims were aggregated.

However, in many applications of aggregate signatures, an attacker might be able to observe partial aggregates, which provide information about the order of aggregation of aggregates at a later point in time.

Furthermore, in practice an attacker could possibly distinguish two order-trees by observing the application and measuring the time it takes to compute

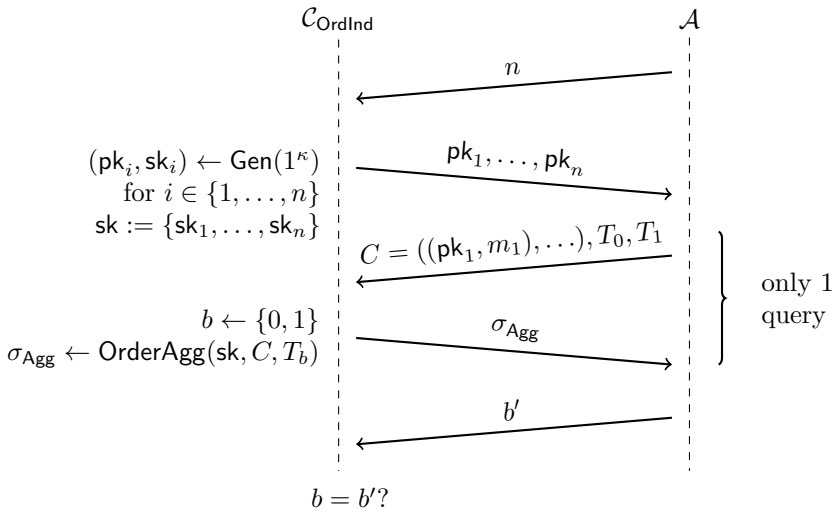

Figure 5.4: The order-independence experiment.

$\mathsf{OrderAgg}(\mathsf{sk}, C, T_b)$, since two different order-trees for the same claim sequence can result in vastly different runtimes. Consider, for example, the trees describing a sequential aggregation and a completely balanced binary tree. Such runtime behavior is usually not modeled in security experiments, since it is assumed that the challenger answers instantly.

Remark. Sequential aggregate signature schemes that satisfy the SAS-EUF-CMA unforgeability definition of [Lys+04] (which we also use as the definition of unforgeability for sequential schemes in this thesis) cannot be order-independent or claim-removable. This definition of unforgeability takes the order of aggregation into account and a reordering of claims is seen as a valid forgery, i.e. the Vfy algorithm needs to be able distinguish different orders of aggregation. Furthermore, if such a scheme would be claim-removable, then the attacker could easily modify prefixes of signatures by removing claims for which he knows the secret keys, which is also interpreted as a valid forgery.

However, other security definitions for sequential schemes that do not preclude claim-removability and order-independence exist, for example the definition of [Lu+06]. See also Section 3.3 for more details on these different definitions.

Example 5.2.20. To illustrate these properties, we now consider the BGLS scheme ([Bon+03], see Section 3.2.1 for a brief overview). Recall that signatures of the BGLS schemes are elements of a cyclic group.

Extandability: Signatures are aggregated by simply multiplying them. BGLS imposes no restriction on aggregation and offers fully flexible aggregation, so every signature can be aggregated with any other and it is always possible to add additional claims. BGLS is therefore extendable.

Claim-Removability: The Sign algorithm is deterministic and for a fixed message m and fixed secret key sk, the returned signature will always

be equal to $H(m)^{\mathsf{sk}}$. Also, aggregate signatures will always be of the form $\prod_j H(m_j)^{\mathsf{sk}_j}$. Suppose σ_{Agg} is an aggregate signature for messages $m_1, \ldots, m_n$ signed under the secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$. Then the owner of sk_i can deaggregate their signature for message m_i by computing

$$\begin{aligned}\frac{\sigma_{\mathsf{Agg}}}{\mathsf{Sign}(\mathsf{sk}_i, m_i)} &= \frac{\prod_{j=1}^{n} H(m_j)^{\mathsf{sk}_j}}{H(m_i)^{\mathsf{sk}_i}} \\ &= \prod_{j=1, j\neq i}^{n} H(m_j)^{\mathsf{sk}_j},\end{aligned}$$

which is a valid aggregate signature for all messages $m_j \neq m_i$. The BGLS scheme is therefore claim-removable.

Order-Independence: Order-independence follows directly from the fact that aggregation is equal to the group operation. Since the group is cyclic, it is also commutative. Furthermore, the scheme is deterministic and the challenger executes all algorithms honestly, so he will always compute the same signature for a given claim (in fact, there even exists only one valid signature for each claim, see Lemma 5.3.1 for a formal proof of this). Therefore, if signatures for the same claims are aggregated, the resulting aggregate signature will always be the same group element, no matter the order.

It follows that the BGLS aggregate signature scheme fulfills all three properties.

Next, we turn our attention to proving that 2DEAGG security implies nDEAGG security for schemes that fulfill these requirements.

Theorem 5.2.21. *Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ be an aggregate signature scheme. If Σ is extendable, claim-removable, order-independent and* 2DEAGG *secure, then it is also* nDEAGG *secure for all $n \geq 2$ polynomial in the security parameter κ.*

Proof. Let $\mathcal{A}$ be a nDEAGG attacker on Σ and $\mathcal{C}_{\mathsf{2DEAGG}}$ be the 2DEAGG challenger. We construct a simulator $\mathcal{B}$ which simulates the nDEAGG security experiment using $\mathcal{C}_{\mathsf{2DEAGG}}$ for $\mathcal{A}$, so that $\mathcal{B}$ has a non-negligible success probability to win against $\mathcal{C}_{\mathsf{2DEAGG}}$ if $\mathcal{A}$ has non-negligible success probability in winning the nDEAGG security experiment. Since we assumed Σ to be 2DEAGG secure, this is a direct contradiction and it therefore follows that it is also nDEAGG secure.

The strategy of the simulator $\mathcal{B}$ is as follow (see Figure 5.5 for a schematic overview of the proof). First, it receives two honestly generated public keys pk' and pk'' from $\mathcal{C}_{\mathsf{2DEAGG}}$. It chooses two distinct indices $v, w \leftarrow \{1, \ldots, n\}$ randomly and sets $\mathsf{pk}_v := \mathsf{pk}'$ and $\mathsf{pk}_w := \mathsf{pk}''$. Next, it generates additional $n-2$ key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^k)$ for all $i \in \{1, \ldots, n\} \setminus \{v, w\}$ and sends all public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ to $\mathcal{A}$. For schemes that only allow the use of distinct keys during aggregation (like [Lu+06], see Section 5.3.6), $\mathcal{B}$ ensures that all keys are distinct.

At some point, $\mathcal{A}$ will continue with the Challenge Phase and send n claims in a claim sequence $C = (c_1, \ldots, c_n) = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$ and an order-tree T. $\mathcal{B}$ ignores T and sends $C_{vw} := ((\mathsf{pk}_v, m_v), (\mathsf{pk}_w, m_w))$ and T_{vw} to $\mathcal{C}_{\mathsf{2DEAGG}}$,

where T_{vw} is the order-tree specifying that the individual signature of m_w should be aggregated to the individual signature of m_v. The challenger will answer with an honestly generated aggregate signature σ_{vw} for C_{vw}.

Since $\mathcal{B}$ knows all secret keys (except for sk_v and sk_w) and Σ is extendable, it can generate individual signatures σ_i for all remaining claims (pk_i, m_i) using sk_i and aggregate them successively to σ_{vw}. The resulting aggregate signature σ_{Agg} is a valid signature for the claim sequence C and $\mathcal{B}$ will send it to $\mathcal{A}$. Observe that $\mathcal{B}$ has completely ignored the order-tree T sent by $\mathcal{A}$, but since Σ is order-independent, $\mathcal{A}$ cannot distinguish this signature from one that was generated by adhering to T, which we discuss further in the probability analysis.

After having received σ_{Agg}, $\mathcal{A}$ will answer with a claim sequence C^* and corresponding signature σ^*. $\mathcal{B}$ will abort if (C^*, σ^*) is not a successful deaggregation answer. If $\mathcal{A}$ successfully deaggregated at least one claim, we have that

$$\mathsf{Vfy}(C^*, \sigma^*) = 1 \wedge |C^*| \leq n - 1 \wedge \forall c_i \in C^* : c_i \in C.$$

We now have that there must be *at least* two distinct claims $c \neq c'$ in C, such that $c \in C^*$, but $c' \notin C^*$. If $c, c' \notin \{c_v, c_w\}$, then $\mathcal{B}$ aborts. Otherwise, without loss of generality let $c = c_v$ and $c' = c_w$. This implies that σ^* contains a valid signature for c_v, but c_w was deaggregated.

Σ is claim-removable and $\mathcal{B}$ knows all secret keys except for sk_v and sk_w, hence $\mathcal{B}$ is able to remove all other claims $c_i \neq c_v$ from σ^* to compute a valid signature σ_v for c_v. $\mathcal{B}$ will send $(\mathsf{pk}_v, m_v), \sigma_v$ to $\mathcal{C}_{\mathsf{2DEAGG}}$ to win the experiment.

Next, we analyze the success probability of $\mathcal{B}$, which is dependent on the success probability of $\mathcal{A}$ and its output. First, we argue that the success probability of $\mathcal{A}$ in the simulation of $\mathcal{B}$ is only negligibly changed, although the simulation is not perfect, since $\mathcal{B}$ did not adhere to the order-tree T sent by $\mathcal{A}$.

If this would not be the case, we could construct an algorithm $\mathcal{D}$ which would break the order-independence of Σ. $\mathcal{D}$ outputs n and then simulates the $n\mathsf{DEAGG}$ experiment for $\mathcal{A}$ using the keys provided by the order-independence challenger. $\mathcal{D}$ sends the claims of $\mathcal{A}$, the order-tree T of $\mathcal{A}$ and the order-tree $T_{\mathcal{B}}$, which is implicitly defined by the aggregation steps of $\mathcal{B}$ in the above described simulation (i.e. first aggregating two randomly chosen claims/signatures and then aggregating the rest successively) to the order-independence challenger. $\mathcal{D}$ outputs 0 if $\mathcal{A}$ successfully deaggregates and 1 otherwise. If the order-independence challenger chooses T, then $\mathcal{D}$ will simulate the normal $n\mathsf{DEAGG}$ experiment. If it chooses $T_{\mathcal{B}}$, then the simulation of $\mathcal{D}$ is equivalent to the simulation of $\mathcal{B}$. Since Σ is order-independent, it follows that the difference in the success probability of $\mathcal{D}$ for both cases is only negligible. Therefore, the success probability of $\mathcal{A}$ in the simulation of $\mathcal{B}$ is only negligibly smaller then in the correct $n\mathsf{DEAGG}$ experiment.

For $\mathcal{B}$ to be successful, the claim sequence C^* output by $\mathcal{A}$ must either contain c_v and not c_w or the other way around. Since C^* must at least contain one claim, the probability that c_v is contained in C^* is at least $1/n$, since the index v was chosen randomly from $\{1, \ldots, n\}$. Moreover, $\mathcal{A}$ has to deaggregate at least one claim, so the probability that c_w is not contained in C^* is also at

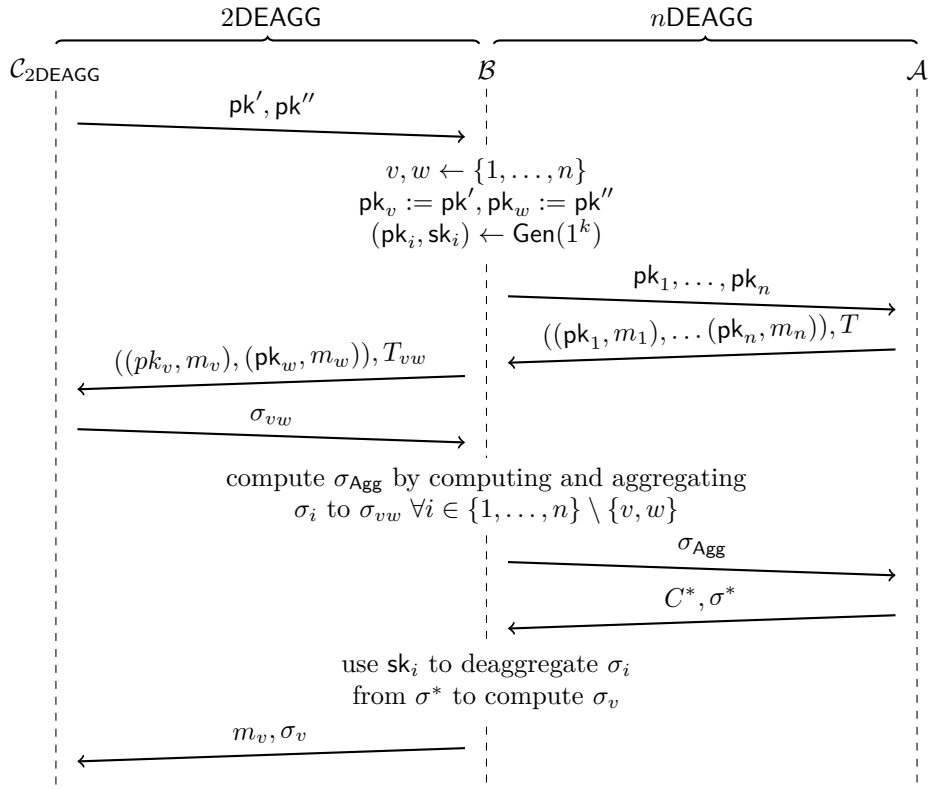

Figure 5.5: Overview of the proof strategy for Theorem 5.2.21.

least $1/n$, because w was also chosen randomly from $\{1, \ldots, n\}$. The situation is analogous for the case that c_v is deaggregated, but c_w is not. All in all we have

$$\begin{aligned}\Pr[\mathcal{B} \text{ is successful}] &\geq \frac{1}{n^2} \cdot \Pr[\mathcal{A} \text{ is successful in the simulation of } \mathcal{B}] \\ &\geq \frac{1}{n^2} \cdot (\Pr[\mathcal{A} \text{ is successful in the } n\mathsf{DEAGG} \text{ exp.}] - \mathsf{negl}(\kappa)),\end{aligned}$$

where negl is a function negligible in the security parameter κ. Now, if the success probability of $\mathcal{A}$ in the nDEAGG security experiment would be non-negligible, then the success probability of $\mathcal{B}$ in the 2DEAGG security experiment would be non-negligible as well. This is a contradiction to the assumed 2DEAGG security of Σ, which concludes the proof. □

Theorem 5.2.22. *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *be an aggregate signature scheme. If* Σ *is extendable, claim-removable, order-independent and* nDEAGG *secure, then it is also* n'DEAGG *secure for all* $n, n' \geq 2$ *polynomial in the security parameter* κ *with* $n' \geq n$.

Proof. The proof is analogous to the proof of Theorem 5.2.21. The only difference is that the strategy of $\mathcal{B}$ needs to be adapted, so that it only creates $n' - n$ key pairs and only removes the signatures and claims corresponding to these keys. □

Theorem 5.2.23. *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *be an aggregate signature scheme. If* Σ *is extendable, order-independent and* $n\mathsf{DEAGG}^+$ *secure, then it is also* $n'\mathsf{DEAGG}^+$ *secure for for all* $n, n' \geq 2$ *polynomial in the security parameter* κ *with* $n' \geq n$.

Proof. The proof is analogous to the proofs of Theorem 5.2.21 and Theorem 5.2.22. Note that in contrast to $n\mathsf{DEAGG}$ security, the scheme does not need to be claim-removable. Since $n\mathsf{DEAGG}^+$ security allows the attacker to use additional claims in its output, $\mathcal{B}$ does not need to deaggregate the signatures signed under the keys chosen by him from the answer of $\mathcal{A}$ before passing on the answer to the challenger. □

Note that Theorem 5.2.21, Theorem 5.2.22 and Theorem 5.2.23 imply that the hierarchies of $n\mathsf{DEAGG}$ and $n\mathsf{DEAGG}^+$ security definitions collapse for schemes that are extendable, claim-removable and order-independent, i.e. if n is the smallest number such that the scheme is $n\mathsf{DEAGG}$, respectively $n\mathsf{DEAGG}^+$, secure, then $n'\mathsf{DEAGG}$, respectively $n'\mathsf{DEAGG}^+$ security automatically follows for $n' \geq n$.

Next, we discuss the general relationship between different "levels" of $n\mathsf{DEAGG}$ security, i.e. the question whether $n\mathsf{DEAGG}$ security for some n also implies $n'\mathsf{DEAGG}$ security for some $n \neq n'$. We show that, unfortunately, this is not the case in general, neither for $n' > n$, nor $n < n'$.

To show this, we construct an aggregate signature scheme $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Agg}', \mathsf{Vfy}')$ as a counter-example which is almost identical to the one of Section 5.2.4. Again, signatures are tuples, but if the number of claims is even, the second entry will simply be set to $\perp$[4], i.e. a signature for an even number of claims is of the form

$$\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, \perp),$$

whereas for an uneven number of claims, the tuple again stores a valid signature for one of the claim sequences used to construct the aggregate signature, i.e. the signature is of the form

$$\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2}).$$

This way, if the base scheme Σ is $n\mathsf{DEAGG}$ secure for all n, then Σ' is $n\mathsf{DEAGG}$ secure for even n, but provably *not* $n\mathsf{DEAGG}$ secure for uneven n. We now define this scheme formally.

Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ be an aggregate signature scheme. Then the aggregate signature scheme Σ' is defined as follows:

$\mathsf{Gen}'(1^\kappa)$: Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ and return $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}'(\mathsf{sk}, m)$: Compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ and return (σ, σ).

[4]Note that we can easily extend our argumentation to the case of schemes with optimal compression. Instead of using $\perp$ for an even number of claims, the Agg' algorithm simply needs to choose a random string that has the same bit length as one signature of Σ. Now, if Σ has optimal compression, so does Σ'. For ease of presentation, we have decided against defining the scheme this way. Strictly speaking, the presented scheme is therefore not an aggregate signature scheme, since the signature size is not constant. However, there is a clear upper bound and the signatures are not increasing in size, but rather fluctuating between two different sizes. Furthermore, this problem can easily be fixed, as described.

$\mathsf{Agg}'(C_1, C_2, \sigma_1, \sigma_2)$: Parse $\sigma_i = (\sigma_{i,1}, \sigma_{i,2})$. Compute

$$\sigma_{\mathsf{Agg},1} \leftarrow \mathsf{Agg}(C_1, C_2, \sigma_{1,1}, \sigma_{2,1})$$
$$\sigma_{\mathsf{Agg},2} \coloneqq \begin{cases} \sigma_{1,1}, & \text{if } |C_1| + |C_2| \text{ is uneven,} \\ \bot, & \text{otherwise,} \end{cases}$$

and return $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2})$.

$\mathsf{Vfy}'(C, \sigma)$: Parse $\sigma = (\sigma_1, \sigma_2)$. Ignore σ_2 and return $\mathsf{Vfy}(C, \sigma_1)$.

Lemma 5.2.24. *Σ' is* AS-EUF-CMA *secure, if Σ is* AS-EUF-CMA *secure.*

Proof. The proof is analogous to the proof of Lemma 5.2.10. □

Remark. Note that Σ' is *not* order-independent and the following lemmas and theorems therefore do *not* violate Theorem 5.2.21, Theorem 5.2.22 or Theorem 5.2.23. To see that Σ' is not order-independent, suppose for example that we have three distinct claims c_i with individual signatures $\sigma_i = (\sigma_{i,1}, \sigma_{i,2})$.

- If we compute $\mathsf{Agg}'((c_1), (c_2), \sigma_1, \sigma_2) = (\mathsf{Agg}((c_1), (c_2), \sigma_{1,1}, \sigma_{2,1}), \bot) =: \sigma_{1\text{-}2}$ and then aggregate σ_3, the resulting signature is

  $$\sigma_{1\text{-}2\text{-}3} \coloneqq \mathsf{Agg}'((c_1, c_2), (c_3), \sigma_{1\text{-}2}, \sigma_3) = (\mathsf{Agg}((c_1, c_2), (c_3), \sigma_{1\text{-}2,1}, \sigma_{3,1}), \sigma_{1\text{-}2,1}).$$

- If we compute $\mathsf{Agg}'((c_2), (c_3), \sigma_2, \sigma_3) = (\mathsf{Agg}((c_2), (c_3), \sigma_{2,1}, \sigma_{3,1}), \bot) =: \sigma_{2\text{-}3}$ and then aggregate σ_1, the resulting signature is

  $$\sigma_{2\text{-}3\text{-}1} = \mathsf{Agg}'((c_2, c_3), (c_1), \sigma_{2\text{-}3}, \sigma_1) = (\mathsf{Agg}((c_2, c_3), (c_1), \sigma_{2\text{-}3,1}, \sigma_{1,1}), \sigma_{2\text{-}3,1}).$$

Now, an order-independence attacker simply needs to check if the second signature in the tuple is valid for the claim sequence (c_1, c_2) to decide which of the two orders were used to compute the aggregate signature.

Lemma 5.2.25. *Let $n > 2$ and $n = 1 \bmod 2$. Then Σ' is* not *n*DEAGG *secure.*

Proof. The proof is analogous to the proof of Lemma 5.2.11. If n is uneven, then the signature given by the challenger in the nDEAGG experiment is a tuple (σ_1, σ_2), such that $\mathsf{Vfy}(C', \sigma_2) = 1$ for a claim sequence C', that is known to $\mathcal{A}$ and that is a true subsequence of C. The deaggregation attacker can therefore simply output C' and $(\sigma_2, \bot)$ to win the experiment. □

Lemma 5.2.26. *Let $n \geq 2$ be polynomial in the security parameter κ and $n = 0 \bmod 2$. Then Σ' is n*DEAGG *secure, if Σ is n*DEAGG *secure.*

Proof. The proof is a straightforward reduction. Let $\mathcal{C}_{n\mathsf{DEAGG}}$ be the nDEAGG challenger for Σ and $\mathcal{A}$ a PPT attacker on the nDEAGG security of Σ'. We construct a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to break the nDEAGG security of Σ.

First, $\mathcal{B}$ receives n public keys from $\mathcal{C}_{n\mathsf{DEAGG}}$, which it passes on to $\mathcal{A}$. Once $\mathcal{A}$ sends its claim sequence C and order-tree T to $\mathcal{B}$, it forwards these to $\mathcal{C}_{n\mathsf{DEAGG}}$. The challenger will then answer with a Σ signature σ for C computed by adhering to the order-tree T. $\mathcal{B}$ now sends $\sigma' = (\sigma, \bot)$ to $\mathcal{A}$, which is a valid and correctly computed Σ' signature for the claim sequence C and order-tree T, because σ was computed correctly and n is even. Once $\mathcal{A}$ outputs its deaggregation C^*, σ^*,

$\mathcal{B}$ parses σ^* as (σ_1^*, σ_2^*). If $\mathcal{A}$ outputs a successful deaggregation, then we have that all claims in C^* must also be in C and at least one claim of C must have been deaggregated and not be present in C^*. Furthermore, σ^* must be a valid Σ' signature for C^*, i.e. it holds that $\mathsf{Vfy}'(C^*, \sigma^*) = 1$, which implies that $\mathsf{Vfy}(C^*, \sigma_1^*) = 1$.

Therefore, $\mathcal{B}$ outputs C^*, σ_1^* as its deaggregation to $\mathcal{C}_{n\mathsf{DEAGG}}$ and wins the experiment.

The simulation of $\mathcal{B}$ for $\mathcal{A}$ is perfect and the success probability of $\mathcal{B}$ is equal to the success probability of $\mathcal{A}$. Its runtime is essentially the same as the runtime of $\mathcal{A}$. It follows that the success probability of $\mathcal{A}$ must be negligible, which concludes the proof. □

All in all, we have that Σ' is $n\mathsf{DEAGG}$ secure for all even n, if Σ is $n\mathsf{DEAGG}$ secure, but $n\mathsf{DEAGG}$ *insecure* for all uneven n.

Theorem 5.2.27. *Let* $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Agg}', \mathsf{Vfy}')$ *be an* AS-EUF-CMA *secure aggregate signature scheme and let* $n, n' \geq 2$ *with* $n \neq n'$*. Then it holds that if* Σ' *is* $n\mathsf{DEAGG}$ *or* $n\mathsf{DEAGG}^+$ *secure, then this does* not *imply that it is also* $n'\mathsf{DEAGG}$ *secure or* $n'\mathsf{DEAGG}^+$ *secure, respectively.*

Proof. The theorem follows from Theorem 5.2.9 and the Lemmas 5.2.24, 5.2.25 and 5.2.26 above. □

As already said, the counter-example works because it is not order-independent. However, if the base scheme Σ is both claim-removable and extendable, then so is Σ'. This shows that order-independence is a necessary requirement to prove the Theorems 5.2.21, 5.2.22 and 5.2.23 (i.e. lower levels of deaggregation security imply higher levels), whereas extendability might be optional for all three theorems and claim-removability might also be optional for Theorem 5.2.21 and Theorem 5.2.22. Still, it is unclear how to adapt our proof strategy to schemes which are not claim-removable and extendable.

Remark. The counter-example can easily be adapted to synchronized and sequential aggregate schemes and Theorem 5.2.27 therefore also applies for these two types of aggregation. For synchronized schemes, the same counter-example can be used. For sequential schemes, it needs to be adapted like the counter-example for sequential schemes in Section 5.2.4. However, every time the aggregation of a new claim results in an even number of claims, the second part of the tuple is set to $\perp$. To be precise, the scheme works as follows:

$\mathsf{Gen}'_{\mathsf{SAS}}(1^\kappa)$: Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}_{\mathsf{SAS}}(1^\kappa)$ and return $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{AggSign}'_{\mathsf{SAS}}(\mathsf{sk}, C, \sigma, m)$: Parse $\sigma = (\sigma_1, \sigma_2)$. Compute

- $\sigma_{\mathsf{Agg},1} \leftarrow \mathsf{AggSign}_{\mathsf{SAS}}(\mathsf{sk}, C, \sigma_1, m)$,
- $\sigma_{\mathsf{Agg},2} \coloneqq \begin{cases} \sigma_1, & \text{if } |C| + 1 \text{ is uneven} \\ \perp, & \text{otherwise} \end{cases}$

and output $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2})$.

$\mathsf{Vfy}'_{\mathsf{SAS}}(C, \sigma)$: Parse $\sigma = (\sigma_1, \sigma_2)$. Ignore σ_2 and return $\mathsf{Vfy}_{\mathsf{SAS}}(C, \sigma_1)$.

This scheme is unforgeable in the sense of SAS-EUF-CMA security, but for an uneven number of claims, deaggregation attacks are trivial. Furthermore it is not order-independent, even if the base scheme is, since for uneven numbers of claims, σ_2 can be used to deduce the claim that was aggregated last. Moreover, if the base scheme is extendable and claim-removable[5], then so is Σ'_{SAS}.

It is an open questions whether nDEAGG security can imply n'DEAGG security for $n' < n$ in certain cases (in general, it does not, see Theorem 5.2.27). It is unclear how one could prove this statement, since a simulator could only use the nDEAGG challenger to answer the query of a n'DEAGG attacker. However, since $n' < n$, the simulator would need to remove some claims from the answer before it could pass it on to the n'DEAGG attacker, who expects a signature on a claim sequence of length n'. This suggests that the simulator would need to break nDEAGG security, which seems paradoxical.

5.2.6 Discussion of nDEAGG and nDEAGG$^+$ Security

All security definitions for deaggregation security presented by us so far, i.e. nDEAGG and nDEAGG$^+$ security, have a shared flaw: The attacker may only send one query to the challenger and has no access to a general Sign oracle. Of course, this is not satisfactory, since in most applications attackers can easily observe signatures or might be able to implement some form of signing oracle by abusing implementation errors, social engineering or by other adversarial means. Our motivations for this restriction are as follows:

Interesting Applications: Although the definitions are comparatively weak, they suffice to construct interesting cryptographic primitives and applications. Already in their seminal paper on aggregate signatures, Boneh et al. [Bon$^+$03] also present a construction of a verifiably encrypted signature scheme. To be able to prove its security, they introduce the *k-Element Aggregate Extraction Problem* and the associated *k-Element Aggregate Extraction Assumption*. The problem and assumption are almost identical to our 2DEAGG security definition, but directly tailored towards the algebraic setting of their aggregate signature scheme. In fact, our definitions can be seen as generalizations of their computational assumption. See Section 5.2.1 for a detailed discussion of the k-Element Aggregate Extraction assumption.

Building a Hierarchy of Definitions: In the research of other cryptographic primitives, building a hierarchy of security definitions of varying strengths has been a successful approach to understanding the needed level of security and then in achieving it. For example, in the case of encryption, researching weaker notions like OW-CPA or IND-CPA has led to successful constructions of IND-CCA secure schemes and a better understanding of security for encryption schemes [NY90; CS98]. Similarly, studying notions like EUF-naCMA for digital signatures and weaker primitives like one-time signatures led to the first fast EUF-CMA secure construction [EGM96]. We are of the opinion that a similar hierarchy for deaggregation security is

[5] We stress once more that schemes that are SAS-EUF-CMA secure in the sense of [Lys$^+$04] can never be order-independent and claim-removable, but other sequential schemes using a different definition of unforgeability like [Lu$^+$06] can be. See also Section 5.3.6.

beneficial as well. For example, Section 5.3 discusses the deaggregation security of several known schemes and this hierarchy allows us to precisely classify the deaggregation security offered by these schemes.

Complexity: Constructing efficient aggregate signature schemes already is a hard problem even without trying to provide a strong form of deaggregation security, as can be seen from the fact that after years of research still no efficient fully flexible scheme is known in the standard model. If we add the additional requirement of deaggregation security on top, the problem becomes even more complex. Furthermore, adding deaggregation security to a scheme will most likely negatively influence its efficiency. It therefore seems advantageous to be able to exactly state which kind of deaggregation security is needed for a given application, just like in the case of security for signatures or encryption, where weaker definitions like one-time signatures, EUF-naCMA or IND-CPA are sufficient for some applications and constructions.

Little Research so far: Up to now, very little research has been done on the deaggregation security of aggregate signature schemes. While technically Boneh et al. [Bon+03] already introduced the problem, research mostly focused on finding new constructions and put little regard on deaggregation security. It seems beneficial to analyze the deaggregation security of existing schemes, although it seems unrealistic to expect that they will fulfill a strong type of security definition, since they were not designed with it in mind. As we discuss in Section 5.3, this speculation turns out to be true and most known schemes only offer limited deaggregation security (nDEAGG security in most cases). Our definitions therefore enable us to clearly state which level of security is offered by these schemes.

Looking at the points made above, we argue that it is worthwhile to study the presented security definitions, even though they do not adequately capture many real-life scenarios. In [FLS12] Fischlin et al. and in [SMD14] Saxena et al. introduce strong security notions for (sequential) aggregate signature schemes that also encompass deaggregation security. The main goal of the definition of Fischlin et al. is history-freeness, meaning that the AggSign algorithm no longer receives the previous messages and public keys in its input, but it also implies a strong form of deaggregation security for sequential schemes. However, their definitions only apply to sequential schemes, since they assume that aggregation is a private operation. Furthermore, as Fischlin et al. note themselves, it is unclear if their strong notion can be fulfilled, which is also why they state a relaxation of their notion. For these reasons, we focus on the definition of Saxena et al. in the following Section 5.2.7.

5.2.7 ADEAGG Security: Security against Adaptive Attacks

One problem that arises once multiple Sign queries are allowed is that the attacker can now aggregate and combine the signatures, which might also give him an advantage in deaggregating. For example, the attacker could simply request individual signatures and aggregate them to compute a "deaggregation" of a signature of a larger claim sequence containing these claims. Furthermore, many known aggregate signature schemes are based on algebraic groups (for

example [Bon+03; AGH10]), signatures are group elements and are aggregated by multiplying them in the group. In these schemes, deaggregation can be as easy as inverting group elements if several signatures are known to the attacker. Consider, for example, the BGLS aggregate signature scheme (see Section 3.2.1 for an overview of the scheme):

Example 5.2.28. Let m_1, m_2 and m_3 be three different messages and $\mathsf{pk}_1 = g^{x_1}, \mathsf{pk}_2 = g^{x_2}$ and $\mathsf{pk}_3 = g^{x_3}$ three different public keys. Then $\sigma = H(m_1)^{x_1} \cdot H(m_2)^{x_2} \cdot H(m_3)^{x_3}$ is a valid signature for the claim sequence $C = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2), (\mathsf{pk}_3, m_3))$. Suppose an attacker gets a hold of a valid signature $\sigma' = H(m_2)^{x_2}$ for m_2 under pk_2. Then he can easily compute a valid signature for the claim sequence $C' = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_3, m_3))$ by computing

$$\sigma \cdot \sigma'^{-1} = \frac{H(m_1)^{x_1} \cdot H(m_2)^{x_2} \cdot H(m_3)^{x_3}}{H(m_2)^{x_2}} = H(m_1)^{x_1} \cdot H(m_3)^{x_3}.$$

So, definitions allowing multiple Sign queries need to keep track of all the signatures the attacker requested and somehow quantify whether he computed a meaningful deaggregation or simply output a trivial aggregation from these signatures. Furthermore, we can no longer speak of a single deaggregation challenge. Instead, any meaningful and non-trivial output of the attacker should be interpreted as a successful deaggregation. To be precise, a signature for any claim sequence that he never requested a signature for should be accepted by the experiment. This captures deaggregation attacks, where the attacker asks for several signatures, computes "subsignatures" of them by deaggregating individual claims and then aggregates them to compute his answer. It also means that such a deaggregation definition will imply unforgeability, in contrast to the definitions so far, where this was provably not the case.

While these requirements seem to be quite complex, Saxena, Misra, and Dhar [SMD14] present a clever definition in which they classify claim sequences into *signable*, *weakly signable* and *not signable* claim sequences by identifying claims with prime numbers. Each claim sequence now also has a corresponding integer that is computed by multiplying the prime numbers associated to the claims contained in the sequence. The prime factorization of the integer of a claim sequence can therefore be used to identify which signatures can be aggregated to construct a signature for this sequence.

Intuitively speaking, *signable* claim sequences are those for which the attacker can compute a signature by simply aggregating signatures he received from the challenger. Claim sequences are called *weakly signable* if a signature for them can be computed by first aggregating and then removing claims from the aggregate by "inverting" some other signatures the attacker received from the challenger, like shown in Example 5.2.28 above. Since not all aggregate signature schemes necessarily allow this "inversion" of the aggregation process, these claim sequences are called *weakly* signable.

Example 5.2.29. Identify the claims (pk_i, m_i) of Example 5.2.28 above with the prime numbers 2, 3 and 5. Let σ be a signature for $C = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2), (\mathsf{pk}_3, m_3))$ and σ' be a signature for $C' = ((\mathsf{pk}_2, m_2))$. The integer associated to C is $2 \cdot 3 \cdot 5 = 30$ and the integer for C' is 3. Let $C'' = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2))$. The integer corresponding to C'' is $2 \cdot 5 = 10$. Suppose σ and σ' are all signatures that the attacker received from the challenger.

Then we say that C' is *not signable*, since there is no way of aggregating σ and σ' to compute a signature for C''. This can also be inferred from the fact that there exist no positive $a, b \in \mathbb{Z}$ such that $10 = 30^a \cdot 3^b$. However, if we allow negative a, b, then we have $10 = 30^1 \cdot 3^{-1}$, which means that a signature for C'' is given by aggregating σ^1 and σ'^{-1}. C'' is therefore *weakly signable*.

We now present a generalized version[6] of the definition of Saxena et al. [SMD14], which we call *adaptive deaggregation security* (abbreviated as ADEAGG). Signable and weakly signable claim sequences are defined formally in Definition 5.2.31 after the definition of the ADEAGG security experiment, since the formal definition of these concepts needs to reference parts of the experiment.

We also use a slightly modified OrderAgg algorithm that now also supports the input of an additional list of claim sequences with corresponding signatures. The order-tree then also specifies when these signatures are supposed to be aggregated to the rest. So,

$$\mathsf{OrderAgg}((\mathsf{sk}_1, \ldots, \mathsf{sk}_n), C, ((C_1, \sigma_1), \ldots, (C_k, \sigma_k)), T)$$

computes signatures for the claims in C and then aggregates them and the signatures σ_j for the claim sequences C_j according to T. See Figure 5.6 for a schematic overview of the ADEAGG security experiment.

Definition 5.2.30 (ADEAGG Security Experiment)**.** *The* ADEAGG security experiment *between an attacker $\mathcal{A}$, a challenger $\mathcal{C}$ and an aggregate signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ consists of three phases as follows:*

Setup Phase. *$\mathcal{A}$ outputs a number $n \in \mathbb{N}$. The challenger $\mathcal{C}$ generates n key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\kappa)$ and gives $\mathsf{PK} := (\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$ to the attacker $\mathcal{A}$. The challenger furthermore initializes a set $CS := \emptyset$, which is used to store the claim sequences for which the attacker asks signatures for.*

Query Phase. *If $\mathcal{A}$ sends a claim sequence C, a list of claim sequences and signatures $((C_1, \sigma_1), \ldots, (C_k, \sigma_k))$ $(k \in \mathbb{N})$, where sk_i is a signature[7] for C_i, and an order-tree T, then $\mathcal{C}$ first checks that each public key of C is in PK.[8] If not, $\mathcal{C}$ answers with $\perp$. Next, $\mathcal{C}$ computes an aggregate signature σ_{Agg} for the claims in C and claim sequences C_i according to the given order-tree by executing*

$$\sigma_{\mathsf{Agg}} \leftarrow \mathsf{OrderAgg}((\mathsf{sk}'_1, \ldots, \mathsf{sk}'_k), C, ((C_1, \sigma_1), \ldots, (C_k, \sigma_k)), T),$$

where sk'_i is the secret key corresponding to the public key of claim i in C. Finally, the challenger adds C to CS. $\mathcal{A}$ may repeat this step at will.

[6] The original definition of [SMD14] is strongly tailored towards their scheme and does not take order-trees and other details into account, that are important for other schemes. See the discussion below Definition 5.2.33 for details on the changes.

[7] Note that σ_i must not necessarily be valid for C_i, which also models attackers that might try to deaggregate by using invalid signatures. However, σ_i of course needs to be in the signature space.

[8] The keys of $C_1, \ldots, C_k$ must not necessarily be in PK, since $\mathcal{A}$ provides the signatures on these sequences. For the same reason, they are also not added to CS.

Output Phase. *At the end of the experiment,* $\mathcal{A}$ *sends a tuple* (C^*, σ^*) *to the challenger* $\mathcal{C}$*. The claim sequence* $C^* = ((\mathsf{pk}_1^*, m_1), \ldots, (\mathsf{pk}_n^*, m_n))$ *may contain public keys not in* PK*. Let* $\mathsf{PK}_{\mathcal{A}} = (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_n^*)$*. Then* $\mathcal{A}$ *is* successful*, if*

1. $\mathsf{Vfy}(C^*, \sigma^*) = 1$,
2. *The intersection* $\mathsf{PK} \cap \mathsf{PK}_{\mathcal{A}}$ *is not empty and*
3. C^* *is* not signable *(see Definition 5.2.31 below).*

Note that the the security experiment requires that the claim sequence output by the attacker is not signable, but weakly signable claim sequences can be used to win the experiment. Therefore, schemes were the inversion of the aggregation process is easy given a signature for a subsequence of claims of another signature, like the BGLS scheme, cannot be ADEAGG secure.[9]

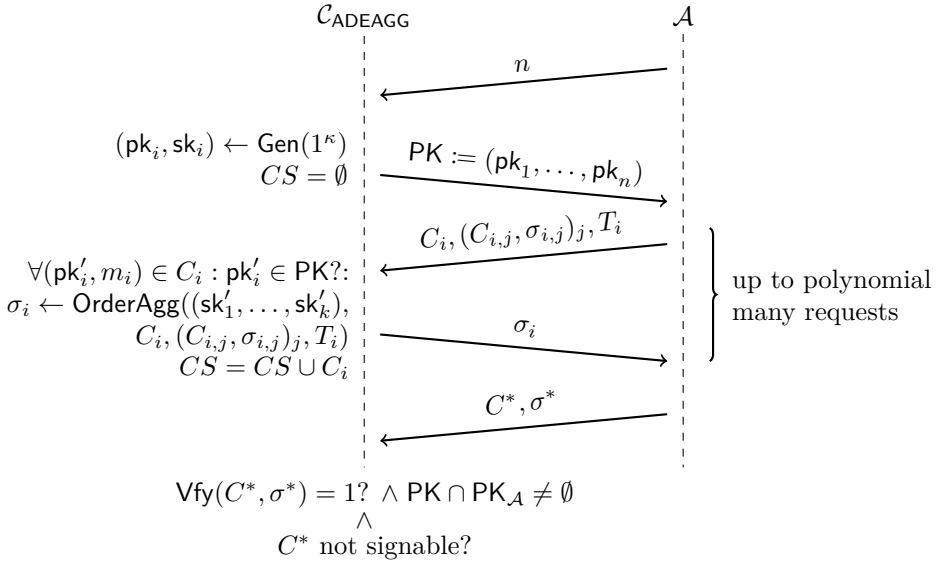

Figure 5.6: The ADEAGG security experiment.

Definition 5.2.31 (Signable and Weakly Signable Claim Sequences)**.** *Let* PK, CS *and* C^* *be as in the* ADEAGG *security experiment above. Let* $C' = \{(\mathsf{pk}, m) \ : \ (\mathsf{pk}, m) \in C^* \wedge \mathsf{pk} \in \mathsf{PK}\}$*. Assign a unique prime number to each element of the set*

$$\{(\mathsf{pk}, m) \ : \ ((\mathsf{pk}, m) \in C \wedge C \in CS) \vee (\mathsf{pk}, m) \in C'\},$$

i.e. the set of all claims which contain a public key chosen by the challenger and which were part of one of the queries of $\mathcal{A}$ *or which were used in the*

[9] To further classify schemes like this, a weaker security definition could be defined that requires that the claim sequence is not weakly signable (and therefore also not signable). However, we will not do so in this thesis.

forgery of $\mathcal{A}$. Then each $C \in CS$ corresponds to a unique integer denoted by $\mathsf{integer}(C)$ *and obtained by multiplying the primes corresponding to its claims. Let* $Z = \{\mathsf{integer}(C) \,:\, C \in CS\}$ *and* $z^* = \mathsf{integer}(C^*)$.

Then the claim sequence C^* is signable *if and only if there exists a solution in* non-negative integers x_i *to the equation*

$$z^* = \prod_{z_i \in Z} z_i^{x_i}.$$

If there exists a solution allowing negative integers, we call C^* weakly signable.

Example 5.2.32. Let m_1, m_2 and m_3 be three different messages. Let $\mathsf{PK} = \{\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4\}$ and $CS = \{C_1, C_2, C_3\}$ with $C_1 = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2))$, $C_2 = ((\mathsf{pk}_2, m_2), (\mathsf{pk}_3, m_3))$, $C_3 = ((\mathsf{pk}_3, m_3), (\mathsf{pk}_4, m_4))$. Let σ_i be a signature for the claim sequence C_i. Let $C^* = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_4, m_4), (\mathsf{pk}_5, m_5))$, where $\mathsf{pk}_5 \notin \mathsf{PK}$ is a public key chosen by the attacker. Assign primes as follows:

$$(\mathsf{pk}_1, m_1) \to 2, \qquad (\mathsf{pk}_2, m_2) \to 3, \qquad (\mathsf{pk}_3, m_3) \to 5, \qquad (\mathsf{pk}_4, m_4) \to 7.$$

Then we have

$$\begin{aligned} Z &= \{\mathsf{integer}(C_1), \mathsf{integer}(C_2), \mathsf{integer}(C_3)\} \\ &= \{2 \cdot 3, 3 \cdot 5, 5 \cdot 7\} \\ &= \{6, 15, 35\} \text{ and} \\ z^* &= \mathsf{integer}(C^*) = 2 \cdot 7 = 14. \end{aligned}$$

Note that the claim (pk_5, m_5) is ignored when assigning primes to the claims and when computing $\mathsf{integer}(C^*)$, since pk_5 is not in PK. $\mathcal{A}$ therefore potentially knows its secret key and can trivially aggregate any claims using this key. We now have that C^* is *weakly signable*, because

$$14 = 6^1 \cdot 15^{-1} \cdot 35^1.$$

However, it is not *signable*, since there are no solutions in non-negative integers to

$$14 = 6^{x_1} \cdot 15^{x_2} \cdot 35^{x_3}.$$

This implies that a signature σ^* for C^* *cannot* be computed by simply aggregating a signature for (pk_5, m_5) and the signatures σ_i. However, if it is possible to "invert" the signatures and the aggregation process (like in the BGLS aggregate signature scheme, see Example 5.2.28), then σ^* could be computed by first aggregating σ_1 to σ_3, removing σ_2 by "aggregating σ_2^{-1}" and then aggregating a signature σ_5 for (pk_5, m_2), i.e. $\sigma^* =$ "$\mathsf{Agg}(\sigma_1, \sigma_2^{-1}, \sigma_3, \sigma_5)$".

Definition 5.2.33 (ADEAGG Security for Aggregate Signature Schemes)**.** *An aggregate signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *is* ADEAGG secure*, if all PPT algorithms $\mathcal{A}$ only have negligible success probability in the* ADEAGG *security experiment, meaning it holds that*

$$\Pr\left[\mathcal{A}^{\mathcal{C}_{\mathsf{ADEAGG}}}(\mathsf{pk}) = (C^*, \sigma^*) : \begin{array}{r} \mathsf{Vfy}(C^*, \sigma^*) = 1 \\ \wedge\, \mathsf{PK} \cap \mathsf{PK}_{\mathcal{A}} \neq \emptyset \\ \wedge\, C^* \textit{is not signable} \end{array}\right] \leq \mathsf{negl}(\kappa)$$

for a function negl*, which is negligible in the security parameter κ.*

Discussion of the Changes to the Original Definition. The original definition of ADEAGG security from [SMD14] is slightly different to the definition given here. We have taken the liberty to adapt and generalize it, so that it fits better to the rest of this thesis and to other aggregate signature schemes than just the original BGLS scheme and the variant of Saxena et al. The changes are as follows:

1. In their work, Saxena et al. do not use claims or claim sequences, but an almost identical concept, which they call *message-descriptor*, the only difference being that a message-descriptor is a *set* of claims.
2. Their definition is geared towards aggregate signature schemes that do not allow to sign the same claim multiple times. We have removed this restriction.
3. Moreover, we have added the possibility for the attacker to specify order-trees in the signing queries. We also allow the attacker to send multiple claim sequences together with corresponding signatures in the queries, so that he can further aggregate claims to signatures that he received from the challenger. Both changes, for example, are necessary to capture security for schemes that are not order-independent or where aggregating the same claims can result in different signatures, even if they are aggregated in the same order.

Although we modified the definition of Saxena et al., these changes do not influence the results presented in [SMD14]. Let $\mathsf{ADEAGG}_{\mathsf{SMD}}$ denote the original definition and let SMD denote their scheme. SMD is based on the the BGLS aggregate signature scheme, which is order-independent (see Example 5.2.20). In fact, aggregating the same individual BGLS signatures will always result in the *same* aggregate signature. A signature of their scheme then is a tuple (σ, R), where R is a list of random strings r_i and σ is a BGLS aggregate signature on the messages $m_i \| r_i \| \mathsf{pk}$, i.e. a random string and the public key get appended to the message before signing.

Because of the order-independence of the BGLS scheme, σ cannot be used to infer the order of aggregation. However, the order of the random strings in the list R is equal to the order of aggregation, so the SMD scheme is not order-independent. But the random strings in R can simply be reordered to create a signature on the same claims in a different order and attackers cannot distinguish such a *reordered* signature from one that was actually computed following this order. In the original definition, it is assumed that the signatures are aggregated in ascending order given by the keys pk_i chosen by the challenger. So if an attacker on the SMD scheme using our definition would request a signature using a different order, we could simply request a signature in ascending order from $\mathcal{C}_{\mathsf{ADEAGG}_{\mathsf{SMD}}}$ and then reorder it accordingly. Therefore, including the possibility that the attacker can specify different order-trees has no repercussions.

Furthermore, for the same reason, the addition that attackers can now also specify claim sequences C_i with signatures σ_i to be aggregated in their queries has no influence either. If an attacker on their scheme would need access to such queries, they could simply be simulated by first requesting a signature for the claims in C from the challenger $\mathcal{C}_{\mathsf{ADEAGG}_{\mathsf{SMD}}}$, then aggregating each σ_i successively, then reordering the random strings according to the given order-tree T and sending this signature to the attacker.

The restriction that the same claim does not appear multiple times still needs to be enforced. But this is already done by the original $\mathsf{Sign}_{\mathsf{SMD}}$ and $\mathsf{Vfy}_{\mathsf{SMD}}$ algorithms anyway. They explicitly check that no claim appears multiple times and abort or output 0 if one does. So, the security experiment does not also need to enforce this restriction.

Therefore, their theorems and statements remain true even for this adapted and generalized definition. For more details on the SMD scheme, see Section 5.3.3.

Remark. Instead of naming their security definition, Saxena et al. call aggregate signature schemes which fulfill it *composite signatures*. However, since we generalized the definition and to better fit the terminology of this thesis, we have chosen to not adapt this term and instead use the name ADEAGG security.

The ADEAGG security definition is strong and captures the possibilities of a real-life adaptive deaggregation attacker quite well, but to the best of our knowledge, there unfortunately exists no ADEAGG secure scheme, except for the SMD scheme. However, the SMD scheme is only secure in the random oracle model and the signature size grows *linearly* in the number of aggregated claims, because for each claim a new random string needs to be added to the signature.

Notice that, in contrast to nDEAGG and $n\mathsf{DEAGG}^+$ security (see Section 5.2.4), ADEAGG security implies unforgeability. Furthermore, it also implies $n\mathsf{DEAGG}^+$ and therefore nDEAGG security:

Theorem 5.2.34. *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *be an aggregate signature scheme. If* Σ *is* ADEAGG *secure, then it is also* $n\mathsf{DEAGG}^+$ *secure for all* $n \geq 2$ *polynomial in the security parameter* κ*.*

Proof. The proof is straightforward. Let $\mathcal{C}_{\mathsf{ADEAGG}}$ be the ADEAGG challenger and $\mathcal{A}$ be a PPT attacker on the $n\mathsf{DEAGG}^+$ security of Σ. We construct a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to win against the challenger $\mathcal{C}_{\mathsf{ADEAGG}}$.

First, $\mathcal{B}$ outputs the number n to $\mathcal{C}_{\mathsf{ADEAGG}}$, who in turn creates n key pairs $(\mathsf{pk}_i, \mathsf{sk}_i)$ and sends $\mathsf{PK} := (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ to $\mathcal{B}$, which $\mathcal{B}$ forwards to $\mathcal{A}$. Once $\mathcal{A}$ outputs its messages $\mathcal{M} = (m_1, \ldots, m_n)$ and order-tree T, $\mathcal{B}$ sets $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$ and sends C together with T to $\mathcal{C}_{\mathsf{ADEAGG}}$. Since all public keys in C are also in PK, the challenger will now correctly compute an aggregate signature σ for C and set $CS := \{C\}$. The simulator $\mathcal{B}$ passes on the signature σ to $\mathcal{A}$. Once $\mathcal{A}$ outputs its deaggregation (C^*, σ^*), the simulator $\mathcal{B}$ simply forwards this tuple to $\mathcal{C}_{\mathsf{ADEAGG}}$. Let $\mathsf{PK}_{\mathcal{A}}$ be the set of public keys in C^*. If the deaggregation attack of $\mathcal{A}$ was successful, we have that

$$\mathsf{Vfy}(C^*, \sigma^*) = 1 \text{ and } \exists c \in C : c \notin C^* \text{ and } \exists c \in C^* : c \in C.$$

This in turn implies that $\mathsf{PK} \cap \mathsf{PK}_{\mathcal{A}} \neq \emptyset$, since there exists at least one claim that is both in C and C^*. Furthermore, since C is the *only* claim sequence in CS and $C \neq C^*$, it also follows that C^* is *not* signable.

So, if $\mathcal{A}$ sends a valid deaggregation, $\mathcal{B}$ also wins its game against $\mathcal{C}_{\mathsf{ADEAGG}}$. The success probability of $\mathcal{B}$ and its runtime are equal to the success probability and runtime of $\mathcal{A}$. Since we assumed the scheme to be ADEAGG secure, it follows that the success probability of $\mathcal{A}$ must be negligible, which concludes the proof. □

The theorem above also directly implies the following corollary:

Corollary 5.2.35. *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ *be an aggregate signature scheme. If* Σ *is* ADEAGG *secure, then it is also* nDEAGG *secure for all* $n \geq 2$ *polynomial in the security parameter* κ.

Limits of ADEAGG Security. Note that ADEAGG security as defined here is not suitable for schemes were signatures are only valid for the exact claim sequence that was used when creating the signature, i.e. were for example a signature for the claim sequence $C = (c_1, c_2)$ is not also valid for the claim sequence $C' = (c_2, c_1)$. The reason is that the multiplication of primes is commutative and therefore we would have $\mathsf{integer}(C) = \mathsf{integer}(C')$. So, if the attacker queries a signature for C, then C' would also wrongly be classified as signable.

Therefore, for these kind of schemes, a more complex way of classifying claim sequences into signable, weakly signable and not signable sequences is needed. However, the definition presented here suffices for all known fully flexible and synchronized aggregate signature schemes, since none of them exhibit the property that the order of claims is critical for verification. The only exception is our fault-tolerant construction from Chapter 4, but since fault-tolerant schemes cannot even be nDEAGG secure (see Section 5.4), no form of ADEAGG security could apply to this construction.

For similar reasons, ADEAGG security only fits fully flexible and synchronized schemes. Since sequential aggregate signatures cannot be aggregated, the concept of signable claim sequences is problematic. For example, suppose an attacker on a sequential scheme asks for signatures σ and σ' for two claim sequences $C = (c_1, c_2)$ and $C' = (c_3, c_4)$. Then the claim sequence $C'' = (c_1, c_2, c_3, c_4)$ would be classified as *signable* by the experiment, although there is no trivial way for the attacker to compute such a signature, since σ and σ' cannot simply be aggregated in a sequential scheme.

A better definition for sequential schemes is given by Fischlin, Lehmann, and Schröder [FLS12]. Their definition is conceptually close to ADEAGG security, but tailor-made for sequential schemes. Their main goal is history-freeness, which means that the AggSign algorithm does not receive the public keys and messages so far in its input. This also implies that the AggSign algorithm can no longer verify the aggregate-so-far, as is done in many sequential schemes. Security needs to be defined differently, since an attacker might request signatures on sequences "starting in the middle" (i.e. where the attacker does not know the corresponding previous messages) or where the given aggregate-so-far is not valid. As a "side effect", these definitions also imply a strong form of deaggregation security for sequential aggregate schemes that is comparable to ADEAGG security. However, Fischlin et al. themselves note that it is unclear whether this definition can be achieved. They also present a relaxed definition and give a scheme based on [Bon+03] that fulfills the relaxed notion in the random oracle model. Furthermore, their definitions do not transfer to fully flexible and synchronized schemes, which they also discuss in detail. For example, the definitions assume that aggregation is a private operation and the aggregation of two signatures is seen as a valid attack. While this is makes sense for sequential schemes, it does not translate to fully flexible and synchronized schemes, where aggregation is supposed to be a publicly computable operation. For these reasons, and also because their scheme is the only known scheme to fulfill their relaxed notion, we do not discuss their definitions in more detail in this thesis.

5.2.8 Adapting the Definitions to Sequential and Synchronized Aggregation

Adapting the definitions presented in the previous chapters to sequential and synchronized aggregate signature schemes is for the most part fairly straightforward. We discuss the details in the following paragraphs.

Sequential Aggregation: Instead of signing each message individually in the experiments and then aggregating the signatures, the messages are sequentially signed and aggregated in the order specified by the attacker. For the $n\mathsf{DEAGG}$ and $n\mathsf{DEAGG}^+$ experiments we no longer need to deal with full order-trees. Instead, sequences of aggregation specified by the attacker suffice. This essentially does not change the definitions. Observe that the attacker has full control over the order in which the messages get signed and aggregated.

The definitions of extendability and claim-removability do not need to be changed. The definition of order-independence in Definition 5.2.19 is changed to work with sequences instead of trees.

Synchronized Aggregation: In the security definitions, if the scheme employs a Setup algorithm, it is executed by the challenger and the attacker is given the public parameters pp at the same time as the public keys. The algorithms of the scheme also receive the synchronizing parameter as additional input during the experiment as needed. The synchronizing parameter is provided by the attacker per query and the same parameter is used for all claims of the query, since only signatures which were created using the same parameter can be aggregated. Concerning the time stamp for $n\mathsf{DEAGG}$ and $n\mathsf{DEAGG}^+$ security, it could be worthwhile to differentiate between definitions that either allow the attacker to use any time stamp in its deaggregation attack or that require him to use the same time stamp as in its query (for the rest of this thesis, we allow the attacker to use any time stamp in his deaggregation answer). For ADEAGG security, the attacker should be allowed use any time stamp, so that the definition remains as general as possible and to still imply unforgeability.

Furthermore, to be in line with the definition of $\mathsf{SyncAS\text{-}EUF\text{-}CMA}$, only one query per time stamp per public key is allowed. Additionally, for all definitions, if the scheme only allows one signature per signer during each time period (like the scheme of Ahn, Green, and Hohenberger [AGH10], see Section 5.3.7), then the challenger ensures that all key pairs chosen in the Setup Phase and given in each query are unique. For ADEAGG security it is additionally necessary that the challenger stores all individual signatures created per query, signer and time period. If the attacker sends another query in the same time period, the challenger needs to check for every claim of the query that it either contains a public key the attacker hasn't used before during this time period or otherwise that the claim is equal to the one stored for this public key in the given time period. If neither is true, then the challenger computes no signature and outputs $\perp$. If it is equal to a previous claim, the challenger needs to use the stored individual signature to compute the aggregate signature for the current query. This way it is ensured that no signer issues more than one signature per time period in the ADEAGG security experiment. Notice that these changes also do not essentially change the definitions and that the attacker still has full control over aggregation.

Since only claims and signatures using the same time stamp can be aggregated, the definition of extendability in Definition 5.2.16 needs to changed so that the claim and signature that are to be added need to be under the same time stamp as the give claim sequence. The definition of order-independence in Definition 5.2.19 also needs to be changed so that all claims use the same time stamp.

All theorems presented in this chapter on deaggregation security also apply to these changed definitions for sequential and synchronized aggregate signature schemes, unless otherwise noted.

5.2.9 Overview of the Deaggregation Security Definitions

To conclude this section about the different deaggregation security definitions, Figure 5.7 gives an overview over the relationships between the definitions. Unforgeability stands representatively for the different definitions of unforgeability for fully flexible, sequential and synchronized aggregate signature schemes, i.e. AS-EUF-CMA, SAS-EUF-CMA and SyncAS-EUF-CMA security. "nDEAGG" and "nDEAGG$^+$" stands representatively for the different levels of nDEAGG, respectively nDEAGG$^+$, security. See the part of the diagram to the right concerning the relationships between these levels.

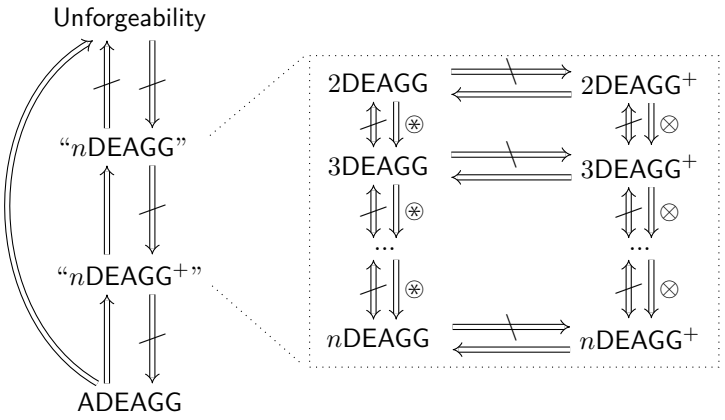

Figure 5.7: Overview of the deaggregation security definitions.

Note that we did not generally prove "nDEAGG $\not\Rightarrow$ nDEAGG$^+$" and "nDEAGG$^+$ $\not\Rightarrow$ ADEAGG". However, in Section 5.3 several schemes are discussed that are nDEAGG, but not nDEAGG$^+$ or nDEAGG$^+$, but not ADEAGG secure (see Theorem 5.3.5 and Theorem 5.3.13, for example).

5.3 Deaggregation Security of Known Schemes

In this section, we discuss the deaggregation security of several known and important fully flexible, sequential and synchronized aggregate signature schemes in detail. Unfortunately, due to the large number of proposed schemes (see the respective overview sections of Chapter 3), we can only discuss the deaggregation security of a limited number of schemes in this thesis.

5.3.1 The BGLS Aggregate Signature Scheme

In this chapter, we discuss the deaggregation security of the BGLS aggregate signature scheme of Boneh, Gentry, Lynn and Shacham [Bon+03] (see Section 3.2.1 for a description of the scheme).

Since the 2EAE assumption is tailor-made for the scheme, it is fairly straightforward to show the 2DEAGG security of BGLS using this assumption. To show this, we also use the following lemma:

Lemma 5.3.1. *Let* $(\mathsf{pk}, \mathsf{sk}) = (g^{\mathsf{sk}}, \mathsf{sk}) \leftarrow \mathsf{Gen}_{\mathsf{BGLS}}(1^k)$ *be* BGLS *key pair (where* g *is a generator of the used group),* m *be a message and* σ *a valid* BGLS *signature for* m *under* pk*. Then* σ *is of the form*

$$\sigma = H(m)^{\mathsf{sk}},$$

i.e. for each message, there exists only one corresponding valid signature.

Proof. Since σ is valid for m under pk, we have $\mathsf{Vfy}_{\mathsf{BGLS}}((\mathsf{pk}, m), \sigma) = 1$. This implies

$$e(\sigma, g) = e(H(m), \mathsf{pk}) = e(H(m), g^{\mathsf{sk}}) = e(H(m)^{\mathsf{sk}}, g).$$

Since $e(\cdot, g)$ is injective (see Lemma 2.2.12), it follows that $\sigma = H(m)^{\mathsf{sk}}$. □

Remark. Note that Lemma 5.3.1 also implies that there exist no irregular but valid signatures for the BGLS scheme.

Theorem 5.3.2. *If the* 2EAE *assumption holds, then the* BGLS *aggregate signature scheme is* 2DEAGG *secure in the random oracle model.*

Proof. Let $\mathcal{A}$ be a 2DEAGG attacker on the BGLS scheme and $\mathcal{C}_{\mathsf{2EAE}}$ the 2EAE challenger. We construct a simulator $\mathcal{B}$ which simulates the 2DEAGG security experiment using $\mathcal{C}_{\mathsf{2EAE}}$ for $\mathcal{A}$ (see Figure 5.8 for a schematic overview of the proof). $\mathcal{B}$ will also control the random oracle and answer all random oracle queries of $\mathcal{A}$. Without loss of generality, we can assume that $\mathcal{A}$ will only send unique random oracle queries.

At the start of the experiment, $\mathcal{B}$ receives g, g^a, g^b, g^u, g^v and g^{au+bv} from the challenger $\mathcal{C}_{\mathsf{2EAE}}$. $\mathcal{B}$ will then pass on $\mathsf{pk}_1 := g^a$ and $\mathsf{pk}_2 := g^b$ to $\mathcal{A}$. Observe that these keys are distributed like honestly created keys, since g^a and g^b were randomly chosen by $\mathcal{C}_{\mathsf{2EAE}}$.

The attacker $\mathcal{A}$ may send a number of random oracle queries before he sends his challenge messages m_1 and m_2 (that we do not know yet at this point in the simulation). We can also assume without loss of generality that $\mathcal{A}$ sends random oracle queries for m_1 and m_2 before he sends the signature query, since m_1 and

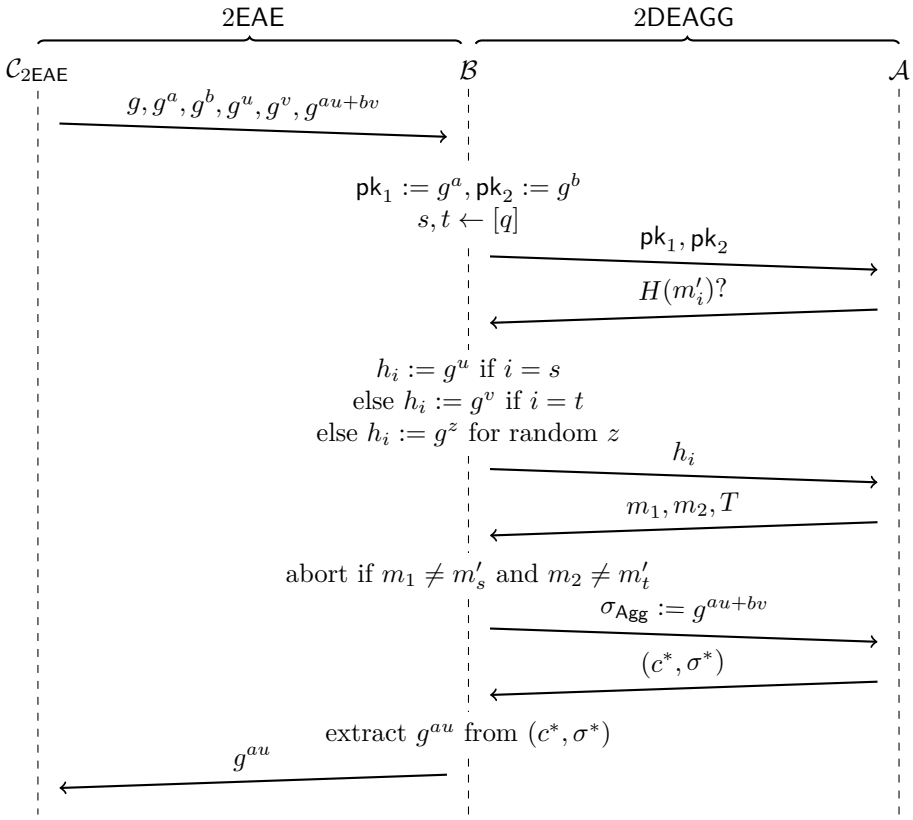

Figure 5.8: Overview of the proof strategy for Theorem 5.3.2.

m_2 will both be signed after $\mathcal{A}$ has sent them, which entails that their random oracle values will be fixed at this point at the latest.

Let $q \in \mathbb{N}$ be the amount of random oracle queries *before* $\mathcal{A}$ sends his challenge messages. $\mathcal{B}$ chooses two distinct indices s, t randomly from $[q]$. Let m'_i be the i-th of these queries. If $i = s$ or $i = t$, then $\mathcal{B}$ will answer with g^u or g^v, respectively. Otherwise $\mathcal{B}$ will pick a random $z \leftarrow \mathbb{Z}_p$ and answer with g^z. Observe that all of these answers are honest answers to the random oracle queries, since g^u and g^v were chosen randomly by $\mathcal{C}_{\mathsf{2EAE}}$.

Once $\mathcal{A}$ sends its two challenge messages m_1 and m_2 with order-tree T, $\mathcal{B}$ continues as follows: First, it checks whether $m_1 = m'_s$ and $m_2 = m'_t$. If this is not the case, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ answers this challenge query by sending $\sigma_{\mathsf{Agg}} := g^{au+bv}$. This is a valid aggregate signature for m_1 and m_2, since individual signature for m_1 and m_2 are equal to $H(m_1)^a = g^{au}$ and $H(m_2)^b = g^{bv}$, according to Lemma 5.3.1, and aggregating them would result in g^{au+bv}. This is also independent of T, since aggregating the same individual signatures will always result in the same group element, no matter in which order they are aggregated.

All random oracle queries that $\mathcal{A}$ sends after the challenge are answered by $\mathcal{B}$ by picking a random $z \leftarrow \mathbb{Z}_p$ and answering with g^z.

If $\mathcal{A}$ is successful, then it sends a claim c^* together with a signature σ^*, such that $\mathsf{Vfy}(c^*, \sigma^*) = 1$ and $c^* = (\mathsf{pk}_1, m_1)$ or $c^* = (\mathsf{pk}_2, m_2)$. If $c^* = (\mathsf{pk}_1, m_1)$, then, according to Lemma 5.3.1, it follows that $\sigma^* = g^{au}$ and $\mathcal{B}$ sends σ^* as his answer to $\mathcal{C}_{\mathsf{2EAE}}$. Otherwise, it holds that $\sigma^* = g^{bv}$ and $\mathcal{B}$ outputs $g^{au+bv} \cdot (\sigma^*)^{-1} = g^{au}$.

Since the probability that $\mathcal{B}$ guessed s and t correctly is at least $1/q$ for both indices, it follows that

$$\Pr[\mathcal{B} \text{ is successful}] \geq \frac{1}{q^2} \cdot \Pr[\mathcal{A} \text{ is successful}],$$

which implies that the success probability of $\mathcal{A}$ must be negligible in the security parameter, because we assumed that the 2EAE problem is hard. □

Theorem 5.3.3. *The* BGLS *aggregate signature scheme is* nDEAGG *secure in the random oracle model under the* 2EAE *assumption for all* $n \geq 2$ *polynomial in the security parameter* κ*.*

Proof. In Example 5.2.20 we have already argued that the BGLS aggregate signature scheme is extendable, claim-removable and order-independent. Therefore, the theorem follows directly from Theorem 5.2.21 and Theorem 5.3.2. □

Corollary 5.3.4. *The* BGLS *aggregate signature scheme is* nDEAGG *secure in the random oracle model under the Computational Diffie-Hellman assumption for all* $n \geq 2$ *polynomial in the security parameter* κ*.*

Proof. This follows directly from Theorem 5.2.2 and Theorem 5.3.3. □

Remark. It should also be possible to directly prove the nDEAGG security of the BGLS scheme under the Computational Diffie-Hellman assumption using techniques similar the ones used by [CN03] to show that the 2EAE and CDH assumptions are equivalent. However, to be consistent with the original publication and formulations of Boneh et al. [Bon+03], we decided to rather prove security under the 2EAE assumption instead and then apply the result of Coron and Naccache [CN03].

Unfortunately, the BGLS aggregate signature scheme does not fulfill the stronger nDEAGG$^+$ security definition. Despite this, a simple change to the algorithms suffices to make the scheme nDEAGG$^+$ secure, as we discuss in Section 5.3.2.

Theorem 5.3.5. *The* BGLS *aggregate signature scheme is* not nDEAGG$^+$ *secure for any* n*.*

Proof. Consider the following PPT attacker $\mathcal{A}$. After $\mathcal{A}$ has received n public keys $\mathsf{pk}_i = g^{a_i}$ from its challenger, it picks n random messages m_i and sends them and a fitting order-tree (for example the order-tree describing sequential aggregation) to the challenger, which computes an aggregate signature σ_{Agg} and sends it to $\mathcal{A}$. $\mathcal{A}$ will then send $C^* = ((\mathsf{pk}_1 \cdot g, m_1), (\mathsf{pk}_2, m_2), \ldots, (\mathsf{pk}_n, m_n))$ and $\sigma^* = \sigma_{\mathsf{Agg}} \cdot H(m_1)$ as its answer. We now have the following:

- $\mathsf{Vfy}(C^*, \sigma^*) = 1$, because

$$\begin{aligned} e(\sigma^*, g) &= e(\sigma_{\mathsf{Agg}} \cdot H(m_1), g) \\ &= e(H(m_1)^{a_1} \cdot \ldots \cdot H(m_n)^{a_n} \cdot H(m_1), g) \\ &= e(H(m_1)^{a_1+1}, g) \cdot e(H(m_2)^{a_2}, g) \cdot \ldots \cdot e(H(m_n)^{a_n}, g) \\ &= e(H(m_1), \mathsf{pk}_1 \cdot g) \cdot e(H(m_2), \mathsf{pk}_2) \cdot \ldots \cdot e(H(m_n), \mathsf{pk}_n), \end{aligned}$$

  so σ^* is a valid signature for the claims in C^*.

- The claim (pk_1, m_1) is not in C^*, because $\mathsf{pk}_1 \neq \mathsf{pk}_1 \cdot g$, so the claim (pk_1, m_1) was successfully deaggregated. All other claims (pk_i, m_i) with $i \neq 1$ are however still in C^*.

The requirements for a successful deaggregation according to the $n\mathsf{DEAGG}^+$ security definition are therefore fulfilled. Furthermore, the runtime of $\mathcal{A}$ is polynomial and its success probability is equal to 1. It follows that the BGLS aggregate signature scheme is *not* $n\mathsf{DEAGG}^+$ secure. □

Remark. In the above attack, deaggregation is done only by changing the public key of the claim of the first message m_1. This might seem counter-intuitive, since one might expect that a successful deaggregation should also remove all traces of the message from the aggregate signature and therefore this deaggregation attack might seem ineffective. We argue that this is not the case.

Since the public key was changed, the parties using the scheme can no longer verify that the party using pk_1 signed the message m_1. Therefore, although m_1 remains in the list of claims, the connection to the first party is severed. Observe that the attacker could also change the claim by multiplying pk_1 with g^z for a random z (and the signature with $H(m_1)^z$), which would result in a completely random public key and would information theoretically hide the connection to pk_1. The definition of $n\mathsf{DEAGG}^+$ security explicitly allows these kinds of attacks to represent this fact. Note that this is also reminiscent of key substitution attacks (see, for example, [MS04; BM99b]).

Theorem 5.3.6. *The* BGLS *scheme is* not ADEAGG *secure.*

Proof. The theorem follows from Theorem 5.2.34 and Theorem 5.3.5 directly above. It also follows from the fact that aggregation can be inverted and an attacker can easily output a weakly signable claim sequence together with a fitting signature, see Example 5.2.28. □

5.3.2 The BNN Aggregate Signature Scheme

Boneh et al. [Bon+03] already propose a simple change to their scheme to fix the problem that only distinct messages can be signed, namely they suggest to sign $H(\mathsf{pk}\|m)$ instead of signing $H(m)$. Still, the $\mathsf{Vfy}_{\mathsf{BGLS}}$ algorithm would need to check that all pairs $\mathsf{pk}\|m$ are distinct, so that their security proof goes through. However, Bellare, Namprempre, and Neven [BNN07] subsequently prove that this check can also be safely removed. Fortunately, this simple change also improves the deaggregation security of the BGLS scheme, since this variant

can be shown to be $n\mathsf{DEAGG}^+$ secure. We call the modified scheme the BNN aggregate signature scheme[10]:

Definition 5.3.7 (BNN Aggregate Signature Scheme)**.** *Let $\mathbb{G}, \mathbb{G}_T$ be cyclic groups of prime order $p \in \mathbb{N}$, g a random generator of $\mathbb{G}$, $e : \mathbb{G}\times\mathbb{G} \to \mathbb{G}_T$ a pairing and $H : \{0,1\}^* \to \mathbb{G}$ a hash function. The* BNN *aggregate signature scheme $\Sigma_{\mathsf{BNN}} = (\mathsf{Gen}_{\mathsf{BNN}}, \mathsf{Sign}_{\mathsf{BNN}}, \mathsf{Agg}_{\mathsf{BNN}}, \mathsf{Vfy}_{\mathsf{BNN}})$ consists of four PPT algorithms as follows:*

$\mathsf{Gen}_{\mathsf{BNN}}(1^\kappa)$**.** *Pick a random $x \leftarrow \mathbb{Z}_p$ and return $(\mathsf{pk}, \mathsf{sk}) := (g^x, x)$.*

$\mathsf{Sign}_{\mathsf{BNN}}(\mathsf{sk}, m)$**.** *Return $\sigma := H(\mathsf{pk}\|m)^{\mathsf{sk}} = H(\mathsf{pk}\|m)^x \in \mathbb{G}$.*

$\mathsf{Agg}_{\mathsf{BNN}}(C_1, C_2, \sigma_1, \sigma_2)$**.** *Return $\sigma_{\mathsf{Agg}} = \sigma_1 \cdot \sigma_2$.*

$\mathsf{Vfy}_{\mathsf{BNN}}(C, \sigma)$*. Parse $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$. If it holds that*

$$e(\sigma, g) = \prod_{i=1}^{n} e(H(\mathsf{pk}_i\|m_i), \mathsf{pk}_i)$$

return 1, otherwise return 0.

Bellare et al. base the security of this scheme on the security of the BGLS aggregate signature scheme (therefore, this scheme is also only proven secure in the random oracle model):

Theorem 5.3.8. *If the* BGLS *aggregate signature scheme is* AS-EUF-CMA *secure, then the* BNN *aggregate signature scheme is also* AS-EUF-CMA *secure.*

Proof. A proof of this theorem can be found in Bellare, Namprempre, and Neven [BNN07]. □

Lemma 5.3.9. *The* BNN *aggregate signature scheme is extendable, claim-removable and order-independent.*

Proof. The proof is analogous to the proof for these properties of the BGLS scheme, see Example 5.2.20. □

Theorem 5.3.10. *If the* 2EAE *assumption holds, then the* BNN *aggregate signature scheme is* $2\mathsf{DEAGG}^+$ *secure in the random oracle model.*

Proof. The proof is in large parts analogous to the proof of Theorem 5.3.2, i.e. the proof that the BGLS scheme is nDEAGG secure. The simulator $\mathcal{B}$ works almost identical and only a few changes are necessary. We therefore omit a few formal details to keep the proof short.

$\mathcal{B}$ receives g, g^a, g^b, g^u, g^v and g^{au+bv} from the challenger $\mathcal{C}_{\mathsf{2EAE}}$. $\mathcal{B}$ will then pass on $\mathsf{pk}_1 := g^a$ and $\mathsf{pk}_2 := g^b$ to $\mathcal{A}$, the supposed $2\mathsf{DEAGG}^+$ attacker on the BNN scheme.

[10] Note that several schemes are discussed in [BNN07]. The one discussed here is called $\mathcal{AS}$-3 in their publication. They also discuss the BGLS scheme and the variant of it that also hashes the public key, but keeps the check for distinct messages in the $\mathsf{Vfy}_{\mathsf{BGLS}}$ algorithm (this variant is also $n\mathsf{DEAGG}^+$ secure, the argumentation is identical to the one presented here). Furthermore, they discuss a slight modification of the LMRS scheme [Lys+04] where the restriction that no public key can be used more than once is removed and the algorithms stay unmodified otherwise. However, this change has no influence on the nDEAGG security of the LMRS scheme (see Section 5.3.4 for details on this scheme).

$\mathcal{B}$ will answer all random oracle queries of $\mathcal{A}$. Without loss of generality, we can assume that $\mathcal{A}$ sends random oracle queries for its messages m_1 and m_2 before sending the aggregation query. Let $q_1, q_2 \in \mathbb{N}$ be the amount of random oracle queries "$\mathsf{pk}\|m$" with $\mathsf{pk} = \mathsf{pk}_1$ or $\mathsf{pk} = \mathsf{pk}_2$, respectively, before $\mathcal{A}$ sends its challenge messages. $\mathcal{B}$ then chooses two random indices $s \leftarrow [q_1], t \leftarrow [q_2]$. For the s-th query that is of the form $\mathsf{pk}_1\|m_s$ for some message m_s, $\mathcal{B}$ answers with g^u. Similarly, $\mathcal{B}$ answers the t-th query of the form $\mathsf{pk}_2\|m_t$ with g^v. For all other queries before and after $\mathcal{A}$ sends the challenge messages, $\mathcal{B}$ chooses random exponents $z \leftarrow \mathbb{Z}_p$ and answers with g^z. All answers are honest answers to the random oracle queries.

Once $\mathcal{A}$ sends its two challenge messages m_1 and m_2 and order-tree T, $\mathcal{B}$ will check whether $m_1 = m_s$ and $m_2 = m_t$. If this is not the case, $\mathcal{B}$ will abort. Otherwise, $\mathcal{B}$ will answer this challenge query by sending $\sigma_{\mathsf{Agg}} := g^{au+bv}$, which is a valid aggregate signature for (pk_1, m_s) and (pk_2, m_t) and all possible order-trees for two messages.

If $\mathcal{A}$ successfully deaggregated, then its output is a claim sequence $C^* = ((g^{x_1}, m_1^*), \ldots, (g^{x_k}, m_k^*))$ of length $k \in \mathbb{N}$ together with a signature σ^*, such that $\mathsf{Vfy}_{\mathsf{BNN}}(C^*, \sigma^*) = 1$ and $\exists c \in ((\mathsf{pk}_1, m_s), (\mathsf{pk}_2, m_t)) : c \notin C^*$ and $\exists c \in C^* : c \in ((\mathsf{pk}_1, m_s), (\mathsf{pk}_2, m_t))$. Without loss of generality[11], we assume that $(\mathsf{pk}_1, m_s) \in C^*$ and $(\mathsf{pk}_2, m_t) \notin C^*$. Let r be the index of the claim (pk_1, m_s) in C^* and let z_i be the random oracle answers to the queries $H(g^{x_i}\|m_i^*)$. Then, since σ^* is valid for C^* we have that

$$\begin{aligned}
\sigma^* &= \prod_{i=1}^{k} H(g^{x_i}\|m_i^*)^{x_i} \\
&= H(\mathsf{pk}_1\|m_s)^a \cdot \prod_{i=1, i\neq r}^{k} H(g^{x_i}\|m_i^*)^{x_i} \\
&= g^{au} \cdot \prod_{i=1, i\neq r}^{k} g^{x_i z_i}.
\end{aligned}$$

Since $\mathcal{B}$ knows all random oracle values z_i and public keys g^{x_i}, he can therefore compute

$$g^{au} = \sigma^* \cdot \left(\prod_{i=1, i\neq r}^{k} (g^{x_i})^{z_i} \right)^{-1}$$

which $\mathcal{B}$ outputs to win the experiment. Since the probabilities that $\mathcal{B}$ guessed s and t correctly are at least $1/q_1$ and $1/q_2$, respectively. It follows that

$$\Pr[\mathcal{B} \text{ is successful}] \geq \frac{1}{q_1 q_2} \cdot \Pr[\mathcal{A} \text{ is successful}],$$

which implies that the success probability of $\mathcal{A}$ must be negligible in the security parameter, because we assumed that the 2EAE problem is hard. □

[11] The argumentation for the case that $(\mathsf{pk}_2, m_2) \in C^*$ is almost identical, except that $\mathcal{B}$ then extracts g^{bv} from σ^* and uses this value to compute g^{au} from g^{au+bv}.

Remark. The crucial difference between the BGLS and BNN schemes that enables us to prove 2DEAGG$^+$ security of the BNN scheme is the addition of the public key in the calculation of the hash values. This ensures that the value g^{bv} will *not* be part of the signature of $\mathcal{A}$ if (pk_2, m_t) is deaggregated, even if $\mathsf{pk}_2 = g^b$ or m_t is reused in some other claim, since $\mathsf{pk}_2 \| m_t$ is the *only* string with the hash value g^v.

Theorem 5.3.11. *The* BNN *aggregate signature scheme is* nDEAGG *secure in the random oracle model under the* 2EAE *assumption for all* $n \geq 2$ *polynomial in the security parameter* κ.

Proof. This follows directly from Theorem 5.2.21, Lemma 5.3.9 and Theorem 5.3.10. □

Corollary 5.3.12. *The* BNN *aggregate signature scheme is* nDEAGG *secure in the random oracle model under the Computational Diffie-Hellman assumption for all* $n \geq 2$ *polynomial in the security parameter* κ.

Proof. This follows directly from Theorem 5.2.2 and Theorem 5.3.11. □

However, the scheme does not satisfy the strong ADEAGG security definition.

Theorem 5.3.13. *The* BNN *aggregate signature scheme is not* ADEAGG *secure.*

Proof. This follows from the fact that aggregation can be inverted by inverting signatures and so an attacker can easily output a weakly signable claim sequence together with a fitting signature. The argumentation is analogous to the argumentation for the BGLS scheme. For this, observe that the BNN scheme is also deterministic and there only exists one valid signature for each claim, just like in the BGLS scheme (see Lemma 5.3.1).

Let m_1 and m_2 be two different messages and $\mathsf{pk}_1 = g^{x_1}$ and $\mathsf{pk}_2 = g^{x_2}$ two different public keys. Then $\sigma = H(\mathsf{pk}_1 \| m_1)^{x_1} \cdot H(\mathsf{pk}_2 \| m_2)^{x_2}$ is the only valid signature for the claim sequence $C = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2))$ and $\sigma' = H(\mathsf{pk}_2 \| m_2)^{x_2}$ is the only valid signature for m_2 under pk_2. Then an ADEAGG attacker can create a valid signature for $C' = ((\mathsf{pk}_1, m_1))$ by asking the challenger for these two signatures and computing

$$\sigma \cdot \sigma'^{-1} = \frac{H(\mathsf{pk}_1 \| m_1)^{x_1} \cdot H(\mathsf{pk}_2 \| m_2)^{x_2}}{H(\mathsf{pk}_2 \| m_2)^{x_2}} = H(\mathsf{pk}_1 \| m_1)^{x_1}.$$

C' is a weakly signable, but *not* signable, claim sequence if the attacker only requests signatures for C and (pk_2, m_2). The attacker therefore wins the ADEAGG experiment with success probability 1. □

5.3.3 The SMD Aggregate Signature Scheme

In [SMD14], Saxena et al. not only define ADEAGG security, but also give a construction of a scheme which is ADEAGG secure in the random oracle model. The scheme is a simple modification of the BGLS scheme and can also be seen as an extension of the BNN scheme: Instead of only hashing the message, they hash the public key, the message and a random value, i.e. they compute a BGLS signature for the string $m \| r \| \mathsf{pk}$. Furthermore, the random value $r \leftarrow \{0,1\}^\kappa$ needs to be stored as a separate value in each individual signature and aggregate

signatures consequently need to store a list of these values. This means that aggregate signatures grow *linearly* in the number of claims. So, the scheme does not compress signatures and therefore fails to achieve one of the central goals of aggregate signatures. However, to the best of our knowledge, it is the only scheme known that fulfills the ADEAGG security definition. The algorithms are as follows:

Definition 5.3.14 (SMD Aggregate Signature Scheme)**.** *The* SMD *aggregate signature scheme* $\Sigma_{\mathsf{SMD}} = (\mathsf{Gen}_{\mathsf{SMD}}, \mathsf{Sign}_{\mathsf{SMD}}, \mathsf{Agg}_{\mathsf{SMD}}, \mathsf{Vfy}_{\mathsf{SMD}})$ *is a variation of the* BGLS *aggregate signature scheme and consists of four PPT algorithms as follows:*

$\mathsf{Gen}_{\mathsf{SMD}}(1^\kappa)$**.** *Pick a random* $x \leftarrow \mathbb{Z}_p$ *an return* $(\mathsf{pk}, \mathsf{sk}) := (g^x, x)$.

$\mathsf{Sign}_{\mathsf{SMD}}(\mathsf{sk}, m)$**.** *Pick a random string* $r \leftarrow \{0,1\}^\kappa$. *Return*

$$\sigma := (H(m \| r \| \mathsf{pk})^{\mathsf{sk}}, r) \in \mathbb{G}.$$

$\mathsf{Agg}_{\mathsf{SMD}}(C_1, C_2, \sigma_1, \sigma_2)$**.** *Check that* $\mathsf{Vfy}(C_i, \sigma_i) = 1$ *and that* $C_1 \cap C_2 = \emptyset$. *Parse* $\sigma_i = (\sigma_i', R_i)$. *Let* R' *be the list containing all strings from* R_1 *and* R_2. *Return*

$$\sigma_{\mathsf{Agg}} = (\sigma_1 \cdot \sigma_2, R').$$

$\mathsf{Vfy}_{\mathsf{SMD}}(C, \sigma)$. *Parse* $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$. *Ensure that all claims are distinct. If not, output* 0. *Otherwise parse* $\sigma = (\sigma', (r_1, r_2, \ldots, r_n))$. *If it holds that*

$$e(\sigma, g) = \prod_{i=1}^{n} e(H(m_i \| r_i \| \mathsf{pk}_i), \mathsf{pk}_i)$$

return 1*, otherwise return* 0.

Unfortunately, the scheme is also only secure in the random oracle model:

Theorem 5.3.15. *If the Computational Diffie-Hellman assumption holds and* H *is modeled as a random oracle, then the* SMD *aggregate signature scheme is* AS-EUF-CMA *and* ADEAGG *secure in the random oracle model.*

Proof. A proof of this theorem can be found in [SMD14]. Note that they prove only ADEAGG security, which however implies AS-EUF-CMA security. □

Corollary 5.3.16. *The* SMD *aggregate signature scheme is* nDEAGG *and* nDEAGG$^+$ *secure in the random oracle model for all* $n \geq 2$ *polynomial in the security parameter* κ *under the Computational Diffie-Hellman assumption.*

Proof. This corollary follows directly from Theorem 5.2.9, Theorem 5.2.34 and Theorem 5.3.15. □

5.3.4 The LMRS Sequential Aggregate Signature Scheme

In [Lys+04], Lysyanskaya, Micali, Reyzin and Shacham introduce the concept of sequential aggregate signatures and present a construction based on trapdoor permutations. Unfortunately, their construction fulfills no notion of deaggregation security, although their definition of unforgeability encompasses a weak form of protection against reordering and removal of claims (see Sections 3.3 and 5.2.4 for details). Their scheme is based on trapdoor permutations:

Definition 5.3.17 (Trapdoor Permutation)**.** *Let* $\mathbb{G}$ *be a group. A* trapdoor permutation family *is a tuple of three PPT algorithms* $\Pi = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ *as follows:*

- $\mathsf{Gen}(1^\kappa)$ *receives as input the security parameter and outputs* (s, t)*, where* s *is a description of a permutation over* $\mathbb{G}$ *and* t *the corresponding trapdoor.*
- $\mathsf{Eval}(s, x)$ *takes as input a description* s *of a permutation over* $\mathbb{G}$ *and a value* $x \in \mathbb{G}$*. It outputs the image of* x *under the permutation described by* s*, i.e.* $s(x)$*, which is also in* $\mathbb{G}$*.*
- $\mathsf{Invert}(s, t, a)$ *takes as input a description* s *of a permutation over* $\mathbb{G}$*, a trapdoor* t *and a value* $a \in \mathbb{G}$*. It outputs a value* $x \in \mathbb{G}$*, such that* $s(x) = a$ *if* t *is a trapdoor corresponding to* s*.*

Definition 5.3.18 (Correctness of Trapdoor Permutations)**.** *A trapdoor permutation family* $\Pi = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ *is* correct*, if for all* $\kappa \in \mathbb{N}$ *and all* $(s, t) \leftarrow \mathsf{Gen}(1^\kappa)$*, it holds that* $\mathsf{Eval}(s, \cdot)$ *is a permutation over* $\mathbb{G}$ *and for all* $x \in \mathbb{G}$ *we have* $\mathsf{Invert}(s, t, \mathsf{Eval}(s, x)) = x$*.*

Definition 5.3.19 (Security of Trapdoor Permutations)**.** *A trapdoor permutation family* $\Pi = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ *is* secure*, if the advantage of any PPT algorithm* $\mathcal{A}$ *in inverting the permutation without knowledge of the trapdoor is negligible, i.e.*

$$\Pr[x \leftarrow \mathcal{A}(s, \mathsf{Eval}(s, x)) \ : \ (s, t) \leftarrow \mathsf{Gen}(1^\kappa), x \leftarrow \mathbb{G}] \leq \mathsf{negl}(\kappa)$$

for a function negl*, which is negligible in the security parameter* κ*.*

Additionally, Lysyanskaya et al. require the trapdoor permutation family to be *certified*, which means that it must be easy to determine if a given s is actually a permutation. Their sequential aggregate signature scheme works as follows:

Definition 5.3.20 (LMRS Sequential Aggregate Signature Scheme)**.** *Let* $\mathbb{G}$ *be a group,* $\Pi = (\mathsf{Gen}_\Pi, \mathsf{Eval}_\Pi, \mathsf{Invert}_\Pi)$ *a certified trapdoor permutation family and* $H : \{0,1\} \to \mathbb{G}$ *a hash function. All users of the scheme need to use the same group, trapdoor permutation family and hash function. The* LMRS *sequential aggregate signature scheme* $\Sigma_{\mathsf{LMRS}} = (\mathsf{Gen}_{\mathsf{LMRS}}, \mathsf{AggSign}_{\mathsf{LMRS}}, \mathsf{Vfy}_{\mathsf{LMRS}})$ *consists of three PPT algorithms as follows:*

$\mathsf{Gen}_{\mathsf{LMRS}}(1^\kappa)$**.** *Pick a random* $(s, t) \leftarrow \mathsf{Gen}_\Pi(1^\kappa)$*. Set* $\mathsf{pk} := s$ *and* $\mathsf{sk} := (s, t)$ *and return* $(\mathsf{pk}, \mathsf{sk})$*.*

$\mathsf{AggSign}_{\mathsf{LMRS}}(\mathsf{sk}, C, \sigma, m)$**.** *Parse* $\mathsf{sk} = (s, t)$*. If* $C = \perp$*, check that* $\sigma = \lambda$*, abort and output* $\perp$ *if not. Otherwise, set* $\sigma = 1$*. Parse* $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$ $(n \in \mathbb{N})$ *and check that* $\mathsf{Vfy}_{\mathsf{LMRS}}(C, \sigma) = 1$*, abort and output* $\perp$ *if not. Otherwise, compute*

$$h \leftarrow H(\mathsf{pk}_1 \| \ldots \| \mathsf{pk}_n \| \mathsf{pk}, m_1 \| \ldots \| m_n \| m) \text{ and}$$
$$\sigma^* \leftarrow \mathsf{Invert}_\Pi(s, t, h \cdot \sigma),$$

where pk *is the public key corresponding to* sk*. Output* σ^* *as the signature.*

$\mathsf{Vfy}_{\mathsf{LMRS}}(C, \sigma)$. *Parse* $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$ $(n \in \mathbb{N})$ *and* $\mathsf{pk}_i = s_i$. *If any* pk_i *appears twice in* C *or if any* s_i *is not a valid permutation*[12]*, output* 0. *Set* $\sigma_n \coloneqq \sigma$. *Then, for* $j = n, \ldots, 1$, *set*

$$\sigma_{j-1} \leftarrow \frac{\mathsf{Eval}_\Pi(s_j, \sigma_j)}{H(\mathsf{pk}_1 \| \ldots \| \mathsf{pk}_j, m_1 \| \ldots \| m_j)}.$$

If $\sigma_0 = 1$, *output* 1, *otherwise output* 0.

Theorem 5.3.21. *The* LMRS *sequential aggregate signature scheme is* SAS-EUF-CMA *secure in the random oracle model, if the used certified trapdoor permutation family is secure.*

Proof. A proof of this theorem can be found in [Lys+04]. [BNN07] show that this theorem also remains true if the check that no public key is used more than once in the $\mathsf{Vfy}_{\mathsf{LMRS}}$ algorithm is removed.[13] □

If s_i^{-1} is the inverse of the permutation s_i, then a sequential aggregate signature of the LMRS scheme is of the form

$$s_n^{-1}(h_n \cdot s_{n-1}^{-1}(h_{n-1} \cdot s_{n-2}^{-1}(\cdots s_2^{-1}(h_2 \cdot s_1^{-1}(h_1)) \cdots)))),$$

where $h_j = H(s_1 \| \ldots \| s_j, m_1 \| \ldots \| m_j)$. So, to verify a signature, the $\mathsf{Vfy}_{\mathsf{LMRS}}$ algorithm consecutively evaluates the permutations in the forward direction and "peels off" all layers until the "center" is reached and only the identity of the group $\mathbb{G}$ remains. This ability to "peel off" signatures is exactly the reason why the scheme is not nDEAGG secure. Attackers can remove signatures starting from the end of the sequence at will, since all information needed to "peel off" a signature is publicly known.

Theorem 5.3.22. *The* LMRS *sequential aggregate signature scheme is* not nDEAGG *secure for any* n.

Proof. Let $\Pi = (\mathsf{Gen}_\Pi, \mathsf{Eval}_\Pi, \mathsf{Invert}_\Pi)$ be the certified trapdoor family and H be the hash function used by the LMRS scheme.

Consider the following PPT attacker $\mathcal{A}$ on the nDEAGG security of the LMRS scheme. First, $\mathcal{A}$ receives n public keys and descriptions of certified trapdoor permutations $\mathsf{pk}_i = s_i$ from the challenger. Let t_i be the corresponding trapdoors. $\mathcal{A}$ then chooses n messages m_i at random, sends them to the challenger together with a fitting order sequence describing the sequential aggregation in ascending order from m_1 to m_n and receives a sequential aggregate signature σ_{Agg} for the claim sequence $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$. According to the definition of $\mathsf{AggSign}_{\mathsf{LMRS}}$, the signature σ_{Agg} is of the form

$$\sigma_{\mathsf{Agg}} = s_n^{-1}(h_n \cdot s_{n-1}^{-1}(h_{n-1} \cdot s_{n-2}^{-1}(\cdots s_2^{-1}(h_2 \cdot s_1^{-1}(h_1)) \cdots)))),$$

[12] Here, it is essential that the used trapdoor permutation family is certified, so that this check can be efficiently computed.

[13] We did not remove this check in our presentation of the scheme to be consistent with the original publication [Lys+04] of the scheme.

where $h_j = H(s_1 \| \ldots \| s_j, m_1 \| \ldots \| m_j)$ and $s_i^{-1}(...)$ is short for $\mathsf{Invert}_\Pi(s_i, t_i, ...)$). $\mathcal{A}$ now computes

$$\begin{aligned}\sigma' &:= \frac{\mathsf{Eval}_\Pi(s_n, \sigma_{\mathsf{Agg}})}{H(s_1 \| \ldots \| s_n, m_1 \| \ldots \| m_n)} \\ &= \frac{H(s_1 \| \ldots \| s_n, m_1 \| \ldots \| m_n) \cdot s_{n-1}^{-1}(h_{n-1} \cdot s_{n-2}^{-1}(\cdots s_2^{-1}(h_2 \cdot s_1^{-1}(h_1)) \cdots)))}{H(s_1 \| \ldots \| s_n, m_1 \| \ldots \| m_n)} \\ &= s_{n-1}^{-1}(h_{n-1} \cdot s_{n-2}^{-1}(\cdots s_2^{-1}(h_2 \cdot s_1^{-1}(h_1)) \cdots))),\end{aligned}$$

which is a valid signature for the claim sequence $C' = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_{n-1}, m_{n-1}))$. Note that $\mathcal{A}$ can easily compute this step, since all values s_i, m_i, σ_{Agg} and algorithms needed are either known to $\mathcal{A}$ or public. Furthermore, note that this step is equal to a single step of $\mathsf{Vfy}_{\mathsf{LMRS}}$. $\mathcal{A}$ outputs C', σ' as its answer to the challenger and wins the experiment with probability 1, since σ' is valid for C' and the claim (pk_n, m_n) was deaggregated. The runtime of $\mathcal{A}$ is polynomial. All this taken together proves the theorem. □

Remark. Note that the concrete instantiation of the trapdoor family is irrelevant for the proof of Theorem 5.3.22, which shows that it is not possible to achieve nDEAGG security for the LMRS scheme by using a specific trapdoor permutation.

5.3.5 The NEV Sequential Aggregate Data Scheme

Neven [Nev08] generalized sequential aggregate signature schemes to *sequential aggregate data*. Here, the AggSign and Vfy algorithms only receive the current message, secret key and signature as input. In particular, they do not receive the previous public keys and messages. The goal of this property is to offer additional savings concerning bandwidth. Consequently the notation used by Neven [Nev08] is slightly different and we would therefore also need to adapt the definitions of deaggregation security to be formally precise. However, since these changes are rather straightforward, we do not give formal definitions for the sake of brevity. Note that $\mathsf{Vfy}_{\mathsf{NEV}}$ outputs the used public keys and signed messages and not just a binary value, i.e. the $\mathsf{Vfy}_{\mathsf{NEV}}$ algorithm is able to reconstruct these values from the signature.

Prerequisites. The scheme uses several different tools, which we will only introduce briefly here. First, like the LMRS scheme in the previous Section 5.3.4, the scheme is based on a family of trapdoor permutations. For security, the family additionally needs to be *claw-free*, meaning that given two permutations π_1, π_2, it must be computationally hard to find a tuple (x, y) such that $\pi_1(x) = \pi_2(y)$. The scheme uses two security parameters κ, ℓ, where $\kappa > \ell$ can be chosen independently by each signer, but ℓ is fixed system-wide. Furthermore, let π be a permutation and D_π its domain. It is assumed that there exists an additive abelian group $\mathbb{G}_\pi \subseteq D_\pi$ such that $|\mathbb{G}_\pi| \geq 2^{\kappa-1}$. Also, there must exist an efficient encoding algorithm $\mathsf{enc}_\pi : \{0,1\}^* \to \{0,1\}^* \times \mathbb{G}_\pi$ that breaks up a message m into a (shorter) message ν and an element $\mu \in \mathbb{G}_\pi$. There must also exist an efficient decoding algorithm $\mathsf{dec}_\pi : \{0,1\}^* \times \mathbb{G}_\pi \to \{0,1\}^*$, that also needs to be injective.[14] Let $H : \{0,1\}^* \to \{0,1\}^\ell$ and $G_\pi : \{0,1\}^\ell \to \mathbb{G}_\pi$ be two

[14] Neven [Nev08] shows how to instantiate these tools based on the RSA and Factoring assumptions.

hash functions, which are modeled as random oracles. Furthermore, let ϵ be the empty string and if $\vec{v} = (v_1, \ldots, v_n)$ is a vector, then let $\vec{v}|a$ denote the vector $(v_1, \ldots, v_n, a)$.

Definition 5.3.23 (NEV Sequential Aggregate Data Scheme)**.** *Let $\mathbb{G}$ be a group, $\Pi = (\mathsf{Gen}_\Pi, \mathsf{Eval}_\Pi, \mathsf{Invert}_\Pi)$ a claw-free trapdoor permutation family and $\mathsf{enc}_\pi, \mathsf{dec}_\pi, H$ and G as described above. The* NEV *sequential aggregate data scheme $\Sigma_{\mathsf{NEV}} = (\mathsf{Gen}_{\mathsf{NEV}}, \mathsf{AggSign}_{\mathsf{NEV}}, \mathsf{Vfy}_{\mathsf{NEV}})$ consists of three PPT algorithms as follows:*

$\mathsf{Gen}_{\mathsf{NEV}}(1^\kappa)$**.** *Pick a random $(s,t) \leftarrow \mathsf{Gen}_\Pi(1^\kappa)$. Let π stand for the permutation defined by $\mathsf{Eval}_\Pi(s,\cdot)$ and π^{-1} be the inverse given by $\mathsf{Invert}_\Pi(s,t,\cdot)$. Set $\mathsf{pk} \coloneqq \pi$ and $\mathsf{sk} \coloneqq \pi^{-1}$ and return $(\mathsf{pk}, \mathsf{sk})$.*

$\mathsf{AggSign}_{\mathsf{NEV}}(\mathsf{sk}_n, \sigma_{n-1}, m_n)$**.** *Parse $\mathsf{sk}_n = \pi_n$. If $n = 1$, then let $\sigma_0 = (\epsilon, \epsilon, 0^\ell)$. Otherwise parse $\sigma_{n-1} = (\vec{\pi}, \nu_{n-1}, X_{n-1}, h_{n-1})$, where $\vec{\pi}$ is a vector of $n-1$ permutations. If $\mathsf{Vfy}_{\mathsf{NEV}}(\sigma_{n-1}) = (\bot, \bot)$, then abort and return $\bot$. Otherwise, do the following:*

$$\begin{aligned}(\mu_n, \nu_n) &\coloneqq \mathsf{enc}_{\pi_n}(m_n \| \nu_{n-1} \| X_{n-1}) \\ h_n &\coloneqq h_{n-1} \oplus H(\vec{\pi}|\pi_n \| m_n \| \nu_{n-1} \| X_{n-1}) \\ g_n &\coloneqq G_{\pi_n}(h_n) \\ X_n &\coloneqq \pi_n^{-1}(g_n + \mu_n)\end{aligned}$$

Return $\sigma_n \coloneqq (\vec{\pi} \| \pi_n, \nu_n, X_n, h_n)$.

$\mathsf{Vfy}_{\mathsf{NEV}}(\sigma)$**.** *Parse $\sigma_n = (\vec{\pi} \| \pi_n, \nu_n, X_n, h_n)$, where $\vec{\pi} = (\pi_1, \ldots, \pi_n)$.*
For $i = n, \ldots, 1$ do
If $|\mathbb{G}_{\pi_i}| < 2^\ell$, then return 0.
$g_i \coloneqq G_{\pi_i}(h_i)$, $\mu_i \coloneqq \pi_i(X_i) - g_i$
$m_i \| \nu_{i-1} \| X_{i-1} \coloneqq \mathsf{dec}_{\pi_i}(m_i, \mu_i)$
$h_{i-1} \coloneqq h_i \oplus H((\pi_1, \ldots, \pi_i) \| m_i \| \nu_{i-1} \| X_{i-1})$
If $(m_0, X_0, h_0) = (\epsilon, \epsilon, 0^\ell)$, then return $(\vec{\pi}, (m_1, \ldots, m_n))$, otherwise return $(\bot, \bot)$.

The unforgeability definition for sequentially aggregate data of Neven [Nev08] is similar to the definition of SAS-EUF-CMA of Lysyanskaya et al. [Lys+04]. We do not formally introduce it and denote it by $\mathsf{SAS\text{-}EUF\text{-}CMA}_{\mathsf{NEV}}$.

Theorem 5.3.24. *The* NEV *sequential aggregate aggregate data scheme is* $\mathsf{SAS\text{-}EUF\text{-}CMA}_{\mathsf{NEV}}$ *secure, if the hash functions H and G_π are modeled as random oracles and the used trapdoor permutation family is claw-free and secure.*

Proof. A proof of this theorem can be found in [Nev08]. □

Just like the LMRS scheme from Section 5.3.4, the NEV scheme is not nDEAGG secure, since the $\mathsf{Vfy}_{\mathsf{NEV}}$ algorithm already describes a deaggregation attack:

Theorem 5.3.25. *The* NEV *sequential aggregate data scheme is* not nDEAGG *secure for any* n.

Proof. Consider the following PPT attacker $\mathcal{A}$ on the nDEAGG security of the NEV scheme. First, $\mathcal{A}$ receives n public keys $\mathsf{pk}_i = \pi_i$ from the challenger. $\mathcal{A}$ then chooses n messages m_i at random, sends them to the challenger together with a fitting order sequence describing the sequential aggregation in ascending order from m_1 to m_n and receives a sequential aggregate signature σ_{Agg} for the claim sequence $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$. According to the definition of $\mathsf{AggSign}_{\mathsf{NEV}}$, the signature σ_{Agg} is of the form

$$\sigma_{\mathsf{Agg}} = ((\pi_1, \ldots, \pi_n), \nu_n, X_n, h_n).$$

$\mathcal{A}$ then computes $g_n \coloneqq G_{\pi_n}(h_n)$ and $\mu_n \coloneqq \pi_n(X_n) - g_n$. These values are then used by $\mathcal{A}$ to compute $m_n \| \nu_{n-1} \| X_{n-1} \coloneqq \mathsf{dec}_{\pi_n}(\nu_n, \mu_n)$ and $h_{n-1} = h_n \oplus H((\pi_1, \ldots, \pi_n) \| m_n \| \nu_{n-1} \| X_{n-1})$. Observe that this is essentially the same as computing one round of the for-loop of $\mathsf{Vfy}_{\mathsf{NEV}}$.

$\mathcal{A}$ then sets $\sigma^* \coloneqq ((\pi_1, \ldots, \pi_{n-1}), \nu_{n-1}, X_{n-1}, h_{n-1})$, which is a valid signature for the claim sequence $C^* \coloneqq ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_{n-1}, m_{n-1}))$. Furthermore, the claim (pk_n, m_n) was deaggregated. $\mathcal{A}$ then outputs C^*, σ^* as its answer to the challenger and wins the experiment with probability 1. Since the runtime of $\mathcal{A}$ is polynomial, this proves the theorem. □

5.3.6 The LOSSW Sequential Aggregate Signature Scheme

Lu et al. [Lu+06] proposed the first sequential aggregate signature scheme that is secure in the standard model. It is based on the digital signature scheme of Waters [Wat05] and makes use of cyclic groups $\mathbb{G}, \mathbb{G}_T$ with a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and prime order p. We discuss its deaggregation security in this section.

Definition 5.3.26 (LOSSW Sequential Aggregate Signature Scheme)**.** *Let* $\mathcal{M} = \{0,1\}^k$ *be the message space for a fixed* $k \in \mathbb{N}$. *The* LOSSW *sequential aggregate signature scheme* $\Sigma_{\mathsf{LOSSW}} = (\mathsf{Gen}_{\mathsf{LOSSW}}, \mathsf{AggSign}_{\mathsf{LOSSW}}, \mathsf{Vfy}_{\mathsf{LOSSW}})$ *consists of three PPT algorithms as follows:*

$\mathsf{Gen}_{\mathsf{LOSSW}}(1^\kappa)$**.** *Pick random* $\alpha, y', y_1, \ldots, y_k$ *from* $\mathbb{Z}_p$. *Compute*

$$\begin{aligned} u' &\coloneqq g^{y'}, \\ u_i &\coloneqq g^{y_i} \text{ for } i \in [k] \text{ and} \\ A &\coloneqq e(g,g)^\alpha. \end{aligned}$$

Set $\mathsf{sk} \coloneqq (\alpha, y', y_1, \ldots, y_k)$ *and* $\mathsf{pk} \coloneqq (A, u', u_1, \ldots, u_k)$ *and output* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{AggSign}_{\mathsf{LOSSW}}(\mathsf{sk}, C, \sigma, m)$**.** *Parse* $\mathsf{sk} = (\alpha, y', y_1, \ldots, y_k)$ *and* $m = (m_1, \ldots, m_k) \in \{0,1\}^k$ *(i.e.* m_i *is the* i*-th bit of* m*). Check that* $\mathsf{Vfy}_{\mathsf{LOSSW}}(C, \sigma) = 1$ *and abort if this is not the case. Next, check whether the public key* pk *corresponding to* sk *already appears in some claim in* C *and abort if it does. Otherwise, parse* $\sigma = (\sigma_1, \sigma_2)$ *and let* $\ell \coloneqq |C|$. *For each* $(\mathsf{pk}_i, m_i) \in C$, *parse* $m_i = (m_{i,1}, \ldots, m_{i,k})$ *and* $\mathsf{pk}_i = (A_i, u_i', u_{i,1}, \ldots, u_{i,k})$. *Compute*

$$\begin{aligned} w_1 &\coloneqq \sigma_1 \cdot g^\alpha \cdot \sigma_2^{y' + \sum_{j=1}^k y_j m_j}, \\ w_2 &\coloneqq \sigma_2. \end{aligned}$$

The values w_1 and w_2 would already form a valid signature, but need to be rerandomized, so that the creator of σ cannot extract g^α from the signature. Therefore, choose $r \leftarrow \mathbb{Z}_p$ and compute

$$\sigma_1' := w_1 \cdot \left(u' \prod_{j=1}^{k} u_j^{m_j} \right)^r \cdot \prod_{i=1}^{\ell} \left(u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}} \right)^r ,$$

$$\sigma_2' := w_2 \cdot g^r .$$

Output $\sigma_{\mathsf{Agg}} = (\sigma_1', \sigma_2')$.

$\mathsf{Vfy}_{\mathsf{LOSSW}}(C, \sigma)$**.** *Check that no public key appears twice in C and output 0 if one does. Otherwise, parse $\sigma = (\sigma_1, \sigma_2)$. Set $\ell = |C|$. If $\ell = 0$, output 1 if $\sigma = \lambda$ and 0 otherwise. For $\ell > 0$, for each $(\mathsf{pk}_i, m_i) \in C$ parse $\mathsf{pk}_i = (A_i, u_i', u_{i,1}, \ldots, u_{i,k})$ and $m_i = (m_{i,1}, \ldots, m_{i,k})$. Then, verify that*

$$e(\sigma_1, g) \cdot e\left(\sigma_2, \prod_{i=1}^{\ell} (u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}}) \right)^{-1} = \prod_{i=1}^{\ell} A_i$$

holds and output 1 if it does and 0 otherwise.

For security, Lu et al. [Lu+06] use a slightly different definition then [Lys+04] (that we presented as the security definition for sequential aggregate signatures in Section 3.3). First, they operate in a certified-key model, where the attacker needs to prove that he knows fitting secret keys to all public keys he is using in his attack (except for the challenge key). Second, the success requirement does not count changes of the prefix before the signature under the challenge public key as valid forgeries and instead is formulated more like the success requirement of AS-EUF-CMA security. See Section 3.3 for details. We write $\mathsf{SAS\text{-}EUF\text{-}CMA}_{\mathsf{LOSSW}}$ to denote their definition of unforgeability.

Theorem 5.3.27. *If the Computational Diffie-Hellman assumption holds, then the* LOSSW *sequential aggregate signature scheme is* $\mathsf{SAS\text{-}EUF\text{-}CMA}_{\mathsf{LOSSW}}$ *secure.*

Proof. A proof for this theorem can be found in [Lu+06]. Note that they reduce the security of the scheme to the security of the digital signature scheme of Waters [Wat05], which in turn was proven secure under the Computational Diffie-Hellman assumption. □

To prove the nDEAGG security of the scheme, we use the following lemma, which states that valid signatures follow the form described by the $\mathsf{AggSign}_{\mathsf{LOSSW}}$ algorithm. This basically means that no signatures exist that are not in the image of $\mathsf{AggSign}_{\mathsf{LOSSW}}$ and that therefore even maliciously created signatures returned by an attacker need to follow this form to be valid.

Lemma 5.3.28. *Let $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$ be a claim sequence using* LOSSW *public keys $\mathsf{pk}_i = (A_i = e(g,g)^{\alpha_i}, u_i', u_{i,1}, \ldots, u_{i,k})$ and let $\sigma_{\mathsf{Agg}} = (\sigma_1, \sigma_2)$ be a valid* LOSSW *sequential aggregate signature on C. Let $\ell = |C|$. Then it holds that σ_1 and σ_2 are of the form:*

$$\sigma_1 = \prod_{i=1}^{\ell} g^{\alpha_i} \cdot (u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}})^r ,$$

$$\sigma_2 = g^r .$$

Proof. Since σ_2 needs to be a group element of $\mathbb{G}$ and g is a generator of $\mathbb{G}$, there exists a $r \in \mathbb{Z}_p$, such that $\sigma_2 = g^r$. Furthermore, we have that σ is valid for C, so it holds that $\mathsf{Vfy}_{\mathsf{LOSSW}}(C, \sigma) = 1$, which implies

$$\begin{aligned} e(\sigma_1, g) &= e\left(\sigma_2, \prod_{i=1}^{\ell}(u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}})\right) \cdot \prod_{i=1}^{\ell} A_i \\ &= e\left(\prod_{i=1}^{\ell}(u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}})^r, g\right) \cdot \prod_{i=1}^{\ell} e(g^{\alpha_i}, g) \\ &= e\left(\prod_{i=1}^{\ell} g^{\alpha_i} \cdot (u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}})^r, g\right) \end{aligned}$$

Since $e(\cdot, g)$ is bijective (see Lemma 2.2.12) it follows that

$$\sigma_1 = \prod_{i=1}^{\ell} g^{\alpha_i} \cdot (u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}})^r,$$

which concludes the proof. □

Remark. Note that Lemma 5.3.28 also implies that there exist no irregular but valid signatures for the LOSSW scheme.

Lemma 5.3.29. *The* LOSSW *sequential aggregate signature scheme is order-independent.*

Proof. The $\mathsf{AggSign}_{\mathsf{LOSSW}}$ algorithm aggregates by multiplying group elements, that are only dependent on random exponents, the current secret key and message, to already existing group elements. Since the group is cyclic and therefore also commutative, the order in which claims get aggregated is of no consequence and the resulting aggregate signature will always be the same group element, if the same random exponents r are used. Since these exponents are always chosen truly at random for every message, they are not influenced by the order of aggregation. Therefore, the order of aggregation is information theoretically hidden and no PPT attacker can have more than negligible success probability in the order-independence experiment. □

Lemma 5.3.30. *The* LOSSW *sequential aggregate signature scheme is extendable.*

Proof. This follows directly from the definition of the algorithms of the LOSSW scheme. The algorithm $\mathsf{AggSign}_{\mathsf{LOSSW}}$ simply performs several group operations and places no restriction on the number of claims that can be aggregated, except for the restriction that no public key may be used more than once. This restriction is however allowed by the definition of extendability, see Definition 5.2.16. □

Lemma 5.3.31. *The* LOSSW *sequential aggregate signature scheme is claim-removable.*

Proof. Let $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$ be a claim sequence with $\mathsf{pk}_i = (A_i = e(g,g)^{\alpha_i}, u'_i, u_{i,1}, \ldots, u_{i,k})$ and let $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, g^r)$ be a valid LOSSW aggregate signature for C. Let $\mathsf{sk}_x = (\alpha_x, y'_x, y_{x,1}, \ldots, y_{x,k})$ be the corresponding secret key to pk_x for $1 \leq x \leq n$. Since σ_{Agg} is valid, according to Lemma 5.3.28, we have that

$$\sigma_{\mathsf{Agg},1} = \prod_{i=1}^{\ell} g^{\alpha_i} \cdot (u'_i \prod_{j=1}^{k} u_{i,j}^{m_{i,j}})^r.$$

Since we know sk_x and g^r, we can compute the value

$$\begin{aligned} \sigma_x &\coloneqq g^{\alpha_x} \cdot (g^r)^{y'_x + \sum_{j=1}^{k} y_{x,k} m_{x,j}} \\ &= g^{\alpha_x} \cdot (u'_x \prod_{j=1}^{k} u_{x,j}^{m_{x,j}})^r \end{aligned}$$

Now, the signature $\sigma'_{\mathsf{Agg}} \coloneqq (\sigma_{\mathsf{Agg},1} \cdot \sigma_x^{-1}, g^r)$ is a valid signature for the claim sequence $C \setminus ((\mathsf{pk}_x, m_x))$, since

$$\sigma_{\mathsf{Agg},1} \cdot \sigma_x^{-1} = \prod_{i=1, i \neq x}^{\ell} g^{\alpha_i} \cdot (u'_i \prod_{j=1}^{k} u_{i,j}^{m_{i,j}})^r,$$

which proves that the LOSSW scheme is claim-removable. □

Remark. Lu et al. [Lu+06] actually already mention this property of their scheme as a means to circumvent the restriction that each signer may only aggregate one claim. If one signer wants to aggregate a second message, he can simply remove his current claim and add a signature over both messages.

Next, we show that the scheme is 2DEAGG secure:

Theorem 5.3.32. *If the Computational Diffie-Hellman assumption holds, then the* LOSSW *sequential aggregate signature scheme is* 2DEAGG *secure.*

Before we prove the theorem, we give a brief overview of the proof strategy. We construct a simulator $\mathcal{B}$ that simulates the 2DEAGG security experiment for an attacker $\mathcal{A}$ on LOSSW and uses its output to solve the Computational Diffie-Hellman problem. To do this, $\mathcal{B}$ sets up the scheme so that an *individual* signature must contain g^{xy} by hiding g^x and g^y in the parameters and public keys. $\mathcal{B}$ needs to do this in such a way that:

- $\mathcal{B}$ is able to extract g^{xy} from the individual signature output by $\mathcal{A}$.
- $\mathcal{B}$ can compute an aggregate signature for both messages, although individual signatures contain g^{xy}.

If both requirements are fulfilled, $\mathcal{B}$ can answer all queries of $\mathcal{A}$ and use its output to solve the Computational Diffie-Hellman problem.

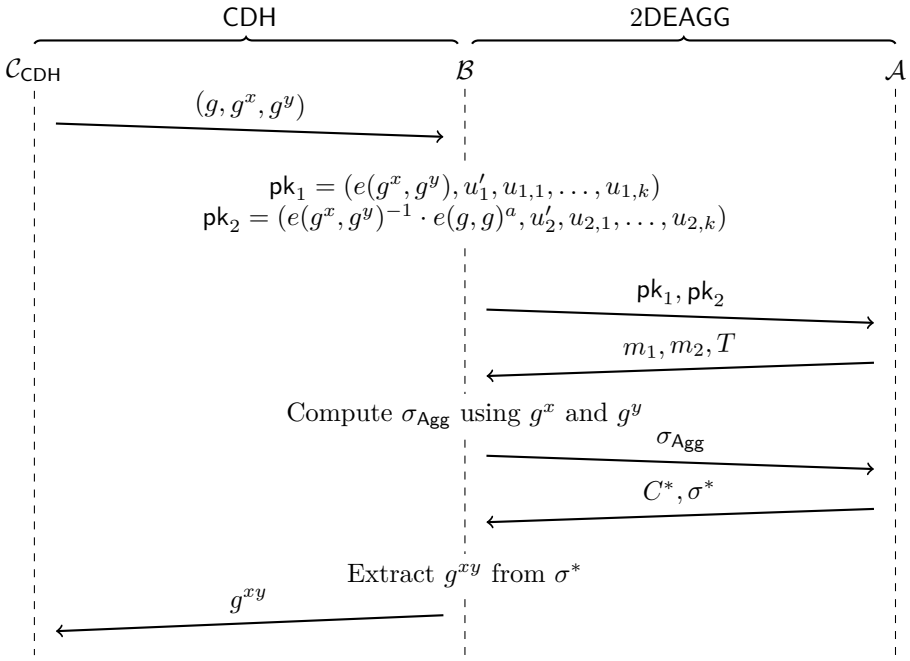

Figure 5.9: Overview of the proof strategy for Theorem 5.3.32.

Proof. Let $\mathcal{C}_{\mathsf{CDH}}$ be the Computational Diffie-Hellman challenger and $\mathcal{A}$ a PPT 2DEAGG attacker on the LOSSW scheme. We construct a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to solve the Computational Diffie-Hellman problem. See Figure 5.9 for a rough overview of the proof strategy.
First, $\mathcal{C}_{\mathsf{CDH}}$ sends a CDH challenge tuple (g, g^x, g^y) to $\mathcal{B}$. Next, $\mathcal{B}$ computes two public keys pk_1 and pk_2 as follows:

- $\mathcal{B}$ picks random $y', z', y_1, \ldots, y_k, z_1, \ldots, z_k$ from $\mathbb{Z}_p$ and for $i \in [k]$ sets

$$\begin{aligned} u'_1 &\coloneqq g^{y'}, \\ u'_2 &\coloneqq g^{z'}, \\ u_{1,i} &\coloneqq g^{y_i}, \\ u_{2,i} &\coloneqq g^{z_i}. \end{aligned}$$

- $\mathcal{B}$ picks a random $a \leftarrow \mathbb{Z}_p$ to compute

$$\begin{aligned} A_1 &\coloneqq e(g^x, g^y) &&= e(g,g)^{xy} \\ A_2 &\coloneqq e(g^x, g^y)^{-1} \cdot e(g,g)^a &&= e(g,g)^{a-xy} \end{aligned}$$

  and sets $\mathsf{pk}_1 \coloneqq (A_1, u'_1, u_{1,1}, \ldots, u_{1,k})$ and $\mathsf{pk}_2 \coloneqq (A_2, u'_2, u_{2,1}, \ldots, u_{2,k})$. Note that $\mathcal{B}$ thereby implicitly sets the α_i elements of both secret keys to $\alpha_1 = xy$ and $\alpha_2 = a - xy$, which $\mathcal{B}$ does not know.

Observe that all elements of both public keys are random group elements, since all exponents are chosen randomly either by $\mathcal{B}$ or by $\mathcal{C}_{\mathsf{CDH}}$ and g is a generator

of $\mathbb{G}$. Furthermore, because $e(g,g)$ is a generator of $\mathbb{G}_T$ and a was also chosen randomly, A_1 and A_2 are also independent of one another. The keys are therefore distributed just like the output of $\mathsf{Gen}_{\mathsf{LOSSW}}$ and $\mathcal{A}$ cannot distinguish them from honestly generated keys. $\mathcal{B}$ now passes pk_1 and pk_2 on to $\mathcal{A}$, who will at some point answer with two messages m_1 and m_2 and an order-tree T.

Next, $\mathcal{B}$ needs to compute an aggregate signature for these two messages. According to Lemma 5.3.28, such a signature for the claim sequence $C := ((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2))$ is of the form $\sigma_{\mathsf{Agg}} = (\sigma_1, \sigma_2)$ with $\sigma_2 = g^r$ for a random $r \in \mathbb{Z}_p$ and

$$\begin{aligned}\sigma_1 &= g^{\alpha_1} \cdot g^{\alpha_2} \cdot (u_1' \prod_{j=1}^{k} u_{1,j}^{m_{1,j}} \cdot u_2' \prod_{j=1}^{k} u_{2,j}^{m_{2,j}})^r, \\ &= g^{xy} \cdot g^{a-xy} \cdot (u_1' \prod_{j=1}^{k} u_{1,j}^{m_{1,j}} \cdot u_2' \prod_{j=1}^{k} u_{2,j}^{m_{2,j}})^r \\ &= g^{a} \cdot (u_1' \prod_{j=1}^{k} u_{1,j}^{m_{1,j}} \cdot u_2' \prod_{j=1}^{k} u_{2,j}^{m_{2,j}})^r,\end{aligned}$$

which is true for any order-tree T, see also the argumentation in Lemma 5.3.29.

Since $\mathcal{B}$ knows a, u_i', $u_{i,1}$, …, $u_{i,k}$ ($i \in \{1,2\}$) and can choose a random r, he can compute such a signature. It is also easy to see that for such a signature $\mathsf{Vfy}_{\mathsf{LOSSW}}(C, \sigma_{\mathsf{Agg}})$ would return 1. $\mathcal{B}$ now computes such a signature and sends it to $\mathcal{A}$.

At some point, $\mathcal{A}$ answers with a claim c^* and a signature $\sigma^* = (\sigma_1^*, \sigma_2^*)$. If $\mathcal{A}$ is successful, we have that $\mathsf{Vfy}_{\mathsf{LOSSW}}(c^*, \sigma^*) = 1$ and without loss of generality $c^* = (\mathsf{pk}_1, m_1)$.[15] Again, according to Lemma 5.3.28, we have that $\sigma_2^* = g^{r^*}$ for some random $r^* \in \mathbb{Z}_p$ and

$$\begin{aligned}\sigma_1^* &= g^{\alpha_1} \cdot (u_1' \prod_{j=1}^{k} u_{1,j}^{m_{1,j}})^{r^*} \\ &= g^{xy} \cdot (\sigma_2^*)^{y' + \sum_{j=1}^{k} y_j m_{1,j}}.\end{aligned}$$

Since $\mathcal{B}$ knows $y', y_1, \ldots, y_k$, he can therefore compute

$$g^{xy} = \sigma_1^* \cdot (\sigma_2^*)^{-(y' + \sum_{j=1}^{k} y_j m_{1,j})},$$

which $\mathcal{B}$ outputs to win the experiment.

$\mathcal{B}$ simulates the 2DEAGG security experiment perfectly for $\mathcal{A}$. The success probability of $\mathcal{B}$ is equal to the success probability of $\mathcal{A}$ and their runtimes are roughly the same (the runtime of $\mathcal{B}$ is basically the run time of $\mathcal{A}$ plus the time it takes to compute the public keys and to extract g^{xy} at the end, which is all polynomial). It follows that the success probability of $\mathcal{A}$ must be negligible, since we assumed that the CDH assumption holds. Therefore the 2DEAGG security of the LOSSW sequential aggregate signature scheme follows. □

[15] If $c^* = (\mathsf{pk}_2, m_2)$, then the rest of the proof is analogous. $\mathcal{B}$ extracts g^{-xy} from σ^*, inverts it and then outputs g^{xy}.

Theorem 5.3.33. *The* LOSSW *sequential aggregate signature scheme is* nDEAGG *secure for all* $n \geq 2$ *polynomial in the security parameter* κ *under the Computational Diffie-Hellman assumption.*

Proof. The theorem follows directly from Theorem 5.2.21, Theorem 5.3.32 and the Lemmas 5.3.29, 5.3.30 and 5.3.31. □

Theorem 5.3.34. *The* LOSSW *sequential aggregate signature scheme is* not nDEAGG$^+$ *secure for any* n.

Proof. Consider the following PPT attacker $\mathcal{A}$ on the nDEAGG$^+$ experiment. First, $\mathcal{A}$ receives n public keys $\mathsf{pk}_i = (A_i = e(g,g)^{\alpha_i}, u_i', u_{i,1}, \ldots, u_{i,k})$ from the challenger. $\mathcal{A}$ chooses n random messages m_i and sends them and a fitting order-tree (for example the order-tree describing sequential aggregation) to the challenger, so that m_i gets signed under pk_i. The challenger answers with a valid signature $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2})$ for the claim sequence $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$, where $\sigma_{\mathsf{Agg},2} = g^r$ for some exponent $r \in \mathbb{Z}_p$.

$\mathcal{A}$ will now manipulate the signature σ_{Agg} so that it becomes valid for a new claim sequence $C' = (c_1', ((\mathsf{pk}_2, m_2), \ldots, (\mathsf{pk}_n, m_n)))$ for a new claim $c_1' = (\mathsf{pk}', m_1) \neq (\mathsf{pk}_1, m_1)$. To do this, $\mathcal{A}$ sets

$$\begin{aligned}\mathsf{pk}' &= (A_1 \cdot e(g,g), u_1', u_{1,1}, \ldots, u_{1,k}) \\ &= (e(g,g)^{\alpha_1+1}, u_1', u_{1,1}, \ldots, u_{1,k}).\end{aligned}$$

Clearly, we have $c_1' \neq (\mathsf{pk}_1, m_1)$ and therefore $C \neq C'$. Since σ_{Agg} is valid, according to Lemma 5.3.28, it holds that

$$\sigma_{\mathsf{Agg},1} = \prod_{i=1}^{\ell} g^{\alpha_i} \cdot (u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}})^r.$$

Next, $\mathcal{A}$ multiplies $\sigma_{\mathsf{Agg},1}$ with g to compute

$$\begin{aligned}\sigma_{\mathsf{Agg},1} \cdot g &= \prod_{i=1}^{\ell} g^{\alpha_i} \cdot (u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}})^r \cdot g \\ &= g^{\alpha_1+1} \cdot (u_1' \prod_{j=1}^{k} u_{1,j}^{m_{1,j}})^r \cdot \prod_{i=2}^{\ell} g^{\alpha_i} \cdot (u_i' \prod_{j=1}^{k} u_{i,j}^{m_{i,j}})^r\end{aligned}$$

and sets $\sigma' = (\sigma_{\mathsf{Agg},1} \cdot g, \sigma_{\mathsf{Agg},2})$, which is a valid signature for the claim sequence C'. $\mathcal{A}$ outputs C' and σ' and wins the experiment. The success probability of $\mathcal{A}$ is equal to 1 and its runtime is polynomial. It follows that the LOSSW scheme is *not* nDEAGG$^+$ secure.

□

The problem and attack strategy is essentially the same as for the BGLS scheme: the attacker changes the public key of one of the claims, but leaves the messages as they were before. Note that, also like in the attack on the BGLS scheme, the attacker could set $\mathsf{pk}' = (e(g,g)^{\alpha_1+r'}, u_1', u_{1,1}, \ldots, u_{1,k})$ for a random exponent $r' \in \mathbb{Z}_p \setminus \{0\}$ to completely hide the connection to pk_1. This also implies that the LOSSW scheme does not fulfill the stronger security notions of [FLS12] for sequential aggregate signature schemes.

5.3.7 The AGH Synchronized Aggregate Signature Scheme

In this section, we discuss the deaggregation security of the synchronized aggregate signature scheme of Ahn, Green, and Hohenberger [AGH10], which is based on the digital signature scheme of Hohenberger and Waters [HW09] and was also the first synchronized scheme to be proven secure in the standard model. The scheme makes use of cyclic groups $\mathbb{G}, \mathbb{G}_T$ with a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, which are chosen by a Setup algorithm. Furthermore, it is assumed that all parties have access to a function $\mathsf{clock}()$ that returns the current time period (which can be any unique integer s in $[1, T]$, where T is some fixed value that is polynomial in the security parameter κ).

Definition 5.3.35 (AGH Synchronized Aggregate Signature Scheme)**.** *The* AGH *synchronized aggregate signature scheme* $\Sigma_{\mathsf{AGH}} = (\mathsf{Setup}_{\mathsf{AGH}}, \mathsf{Gen}_{\mathsf{AGH}}, \mathsf{Sign}_{\mathsf{AGH}}, \mathsf{Agg}_{\mathsf{AGH}}, \mathsf{Vfy}_{\mathsf{AGH}})$ *consists of five PPT algorithms as follows:*

$\mathsf{Setup}_{\mathsf{AGH}}(1^\kappa)$**.** *Select a bilinear group* $\mathbb{G}$ *of prime order* $p > 2^\kappa$*. Let* $Z \in \mathcal{O}(\kappa)$ *be the number of bits in the message space. Let* ℓ, k *be two additional parameters*[16] *such that* $\ell \cdot k = Z$*. These parameters will be used to split up the messages in* k *blocks of* ℓ *bits each. Choose random elements* $g, u_0, \ldots, u_k, w, z, h \in \mathbb{G}$*. Output the public parameters*

$$\mathsf{pp} = (\ell, k, p, \mathbb{G}, g, u_0, \ldots, u_k, w, z, h).$$

$\mathsf{Gen}_{\mathsf{AGH}}(1^\kappa, \mathsf{pp})$**.** *Pick a random* $a \leftarrow \mathbb{Z}_p$*. Set* $\mathsf{pk} := (\mathsf{pp}, g^a)$ *and* $\mathsf{sk} := (\mathsf{pp}, a)$*, initialize* s_{prev} *to zero and return* $(\mathsf{pk}, \mathsf{sk})$*. The parameter* s_{prev} *will be used as an internal state denoting the last time period on which this party issued a signature. Output* $(\mathsf{pk}, \mathsf{sk})$*.*

$\mathsf{Sign}_{\mathsf{AGH}}(\mathsf{sk}, m, s)$**.** *Abort, if* $s = s_{\mathsf{prev}}$ *or* $s \geq 2^\kappa$*. Otherwise set* $s_{\mathsf{prev}} := s$ *(it is assumed that the input* s *is the current time period as output by* $\mathsf{clock}()$*). Let* $m = m_1 \| m_2 \| \ldots \| m_k$*, where each block* m_i *is* ℓ *bits long. Select a random* $t \in \mathbb{Z}_p$*, compute*

$$\sigma_1 := \left(u_0 \prod_{i=1}^{k} u_i^{m_i} \right)^a \cdot (w^{\lceil \log(s) \rceil} z^s h)^t,$$
$$\sigma_2 := g^t,$$

and return $\sigma = (\sigma_1, \sigma_2, s)$ *as the signature.*

$\mathsf{Agg}_{\mathsf{AGH}}(C_1, C_2, \sigma_1, \sigma_2)$**.** *Parse* σ_i *as* $(\sigma_{i,1}, \sigma_{i,2}, s_i)$ *and check that* $s_1 = s_2$*. If this check fails, output* $\perp$*. Otherwise compute*

$$\sigma_{\mathsf{Agg},1} := \sigma_{1,1} \cdot \sigma_{2,1},$$
$$\sigma_{\mathsf{Agg},2} := \sigma_{1,2} \cdot \sigma_{2,2},$$

and output $\sigma_{\mathsf{Agg}} := (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2}, s_1)$*.*

[16] There is no strict rule on how to choose the parameters ℓ and k, but they have direct effects on the security and efficiency of the scheme. For example, the parameter k determines the size of the public parameters. Ahn, Green, and Hohenberger [AGH10] discuss this in detail and suggest using a collision-resistant hash function to obtain 160-bit messages and then setting $k = 5$ and $\ell = 32$.

$\mathsf{Vfy}_{\mathsf{AGH}}(C, \sigma)$. *Parse* $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$ *and* $\sigma = (\sigma_1, \sigma_2, s)$. *Check that* $0 < s < 2^\kappa$. *If this is not the case, abort and output 0. Otherwise let* $m_i = m_{i,1} \| m_{i,2} \| \ldots \| m_{i,k}$, *where each block has* ℓ *bits. Extract* g^{a_i} *from each* pk_i *and compute*

$$V \coloneqq e\left(\prod_{i=1}^{n} g^{a_i}, u_0\right) \cdot \prod_{j=1}^{k} e\left(\prod_{i=1}^{n} g^{a_i m_{i,j}}, u_j\right).$$

Then, verify that

$$e(g, \sigma_1) = V \cdot e(\sigma_2, w^{\lceil \log(s) \rceil} z^s h).$$

Output 1 *if this check is successful, otherwise output* 0.

Note that, just like in the BGLS aggregate signature scheme, the Agg algorithm can easily be extended to aggregate several signatures at the same time. Simply check that all time stamps are equal and then multiply the respective signature parts.

The scheme is restricted so that each signer can issue at most one signature per time period and keeps a state to ensure this (alternatively, additional checks could be added to the algorithms). Once (possibly already aggregated) signatures for two different messages are known, an SyncAS-EUF-CMA attacker can use them to output a valid forgery. Suppose, for example, that signatures σ_1, σ_2 on (pk, m) and $(\mathsf{pk}, 0^Z)$ for $\mathsf{pk} = (\mathsf{pp}, g^a)$ are known. Then we would have

$$\begin{aligned} \sigma_{1,1} &= \left(u_0 \prod_{i=1}^{k} u_i^{m_i}\right)^a \cdot (w^{\lceil \log(s) \rceil} z^s h)^{t_1}, \\ \sigma_{2,1} &= u_0^a \cdot (w^{\lceil \log(s) \rceil} z^s h)^{t_2}. \end{aligned}$$

Set $\sigma' \coloneqq \sigma_{1,1} \cdot \sigma_{2,1}^{-1} = \left(\prod_{i=1}^{k} u_i^{m_i}\right)^a \cdot (w^{\lceil \log(s) \rceil} z^s h)^{t_1 - t_2}$. Then

$$\sigma^* = (\sigma_{1,1} \cdot \sigma', \sigma_{1,2}^2 \cdot \sigma_{2,2}^{-1})$$

is a valid signature for the message $(m_1 + m_1) \| \ldots \| (m_k + m_k)$, since

$$\begin{aligned} \sigma_{1,1} \cdot \sigma^* &= \left(u_0 \prod_{i=1}^{k} u_i^{m_i + m_i}\right)^a \cdot (w^{\lceil \log(s) \rceil} z^s h)^{2t_1 - t_2} \\ \sigma_{1,2}^2 \cdot \sigma_{2,2}^{-1} &= g^{2t_1 - t_2} \end{aligned}$$

Note that in the following Lemma 5.3.40 we show that the scheme is claim-removable. One could therefore assume that in practice, signers could retract their signature and simply aggregate a new one, like in the scheme of [Lu+06]. While this is technically possible, great care has to be taken so that the above mentioned attack cannot be exploited.

Theorem 5.3.36. *The* AGH *synchronized aggregate signature scheme is* SyncAS-EUF-CMA *secure under the Computational Diffie-Hellman assumption.*

Proof. A proof of this theorem can be found in [AGH10]. □

Before we prove the nDEAGG security of the AGH scheme, we first prove the following technical lemma, which states that all valid AGH signatures follow the form described by the $\mathsf{Sign}_{\mathsf{AGH}}$ algorithm. This basically means that no signatures exist that are not in the image of $\mathsf{Sign}_{\mathsf{AGH}}$ and that therefore even maliciously created signatures returned by an attacker need to follow this form to be valid.

Lemma 5.3.37. *Let* $\mathsf{pp} = (\ell, k, p, \mathbb{G}, g, u_0, \ldots, u_k, w, z, h) \leftarrow \mathsf{Setup}_{\mathsf{AGH}}(1^\kappa)$ *be public parameters of the* AGH *scheme and* $\mathsf{pk} = (\mathsf{pp}, g^a)$ *an* AGH *public key. Let* $m = m_1 \| m_2 \| \ldots \| m_k$ *be some message, where each block* m_i *is* ℓ *bits long and let* $\sigma = (\sigma_1, \sigma_2, s)$ *be a valid signature for* m *under* pk*. Then it holds that* σ_1 *and* σ_2 *are of the form*

$$\begin{aligned}\sigma_1 &= \left(u_0 \prod_{i=1}^{k} u_i^{m_i}\right)^a \cdot (w^{\lceil \log(s) \rceil} z^s h)^t, \\ \sigma_2 &= g^t,\end{aligned}$$

for some $t \in \mathbb{Z}_p$*.*

Proof. Since σ_2 needs to be a group element of $\mathbb{G}$ and g is a generator of $\mathbb{G}$, there exists a $t \in \mathbb{Z}_p$, such that $\sigma_2 = g^t$. Furthermore, we have that σ is valid for m, so it holds that $\mathsf{Vfy}_{\mathsf{AGH}}((\mathsf{pk}, m), \sigma) = 1$, which implies that

$$\begin{aligned}e(g, \sigma_1) &= e(g^a, u_0) \cdot e\left(\prod_{i=1}^{k} g^{a m_i}, u_i\right) \cdot e(\sigma_2, w^{\lceil \log(s) \rceil} z^s h) \\ &= e\left(g, \left(u_0 \prod_{i=1}^{k} u_i^{m_i}\right)^a\right) \cdot e(g, (w^{\lceil \log(s) \rceil} z^s h)^t) \\ &= e\left(g, \left(u_0 \prod_{i=1}^{k} u_i^{m_i}\right)^a \cdot (w^{\lceil \log(s) \rceil} z^s h)^t\right).\end{aligned}$$

Since $e(g, \cdot)$ is bijective (see Lemma 2.2.12) it follows that

$$\sigma_1 = \left(u_0 \prod_{i=1}^{k} u_i^{m_i}\right)^a \cdot (w^{\lceil \log(s) \rceil} z^s h)^t,$$

which concludes the proof. □

Remark. Note that Lemma 5.3.37 also implies that there exist no irregular but valid signatures for the AGH scheme.

Lemma 5.3.38. *The* AGH *scheme is order-independent.*

Proof. Let $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_n, \mathsf{sk}_n)$ be n AGH key pairs that were honestly generated by $\mathsf{Gen}_{\mathsf{AGH}}(1^\kappa)$. If $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2}, s)$ is an honestly generated AGH aggregate signature for messages $m_1, \ldots, m_n$ (where each m_i is signed under pk_i), then σ_{Agg} is of the form

$$\sigma_{\mathsf{Agg}} = (\prod_{i=1}^{n} \sigma_{i,1}, \prod_{i=1}^{n} g^{t_i}, s),$$

where $\sigma_i = (\sigma_{i,1}, g^{t_i}, s)$ is a valid individual signature for m_i under pk_i and time stamp s. Aggregation is done by multiplying the parts of these individual signatures. Since the group is cyclic and therefore also commutative, it follows that the order in which these individual signatures get aggregated is of no consequence and $\sigma_{\mathsf{Agg},1}$ and $\sigma_{\mathsf{Agg},2}$ will always be the same group elements, if the same random exponents t_i are used. Since the exponents are chosen truly at random when creating the individual signatures, they are not affected by the order of aggregation. Therefore, the order in which the individual signatures get aggregated is information theoretically hidden and no PPT attacker can have more than negligible success probability in the order-independence experiment. □

Lemma 5.3.39. *The* AGH *scheme is extendable.*

Proof. This follows directly from the definition of the algorithms of the AGH scheme. The aggregation algorithm $\mathsf{Agg}_{\mathsf{AGH}}$ simply performs two multiplications in the group, places no restriction on aggregation (apart from the usual restriction of synchronized aggregate signature schemes that only signatures using the same time period may be aggregated, which is also represented in the respective security definitions) and offers fully flexible aggregation for each time period. □

Lemma 5.3.40. *The* AGH *scheme is extractable.*

Proof. Let $\mathsf{pp} = (\ell, k, \mathbb{G}, g, u_0, \ldots, u_k, w, z, h)$ be the public parameters, $C = ((\mathsf{pk}_1 = (\mathsf{pp}, g^{a_1}), m_1), \ldots, (\mathsf{pk}_n = (\mathsf{pp}, g^{a_n}), m_n))$ a claim sequence and $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, g^t, s)$ a valid AGH aggregate signature for C. For each i we have that $m_i = m_{i,1} \| \ldots \| m_{i,k}$ consists of k blocks of ℓ bits each. Let $\mathsf{sk}_x = (\mathsf{pp}, a_x)$ be the corresponding secret key to pk_x for $1 \leq x \leq n$. Since σ_{Agg} is valid, we have that

$$\begin{aligned} e(g, \sigma_{\mathsf{Agg},1}) &= e\left(\prod_{i=1}^{n} g^{a_i}, u_0\right) \cdot \prod_{j=1}^{k} e\left(\prod_{i=1}^{n} g^{a_i m_{i,j}}, u_j\right) \cdot e(\sigma_{\mathsf{Agg},2}, w^{\lceil \log(s) \rceil} z^s h) \\ &= e\left(g, \prod_{i=1}^{n} \left(u_0 \prod_{j=1}^{k} u_j^{m_{i,j}}\right)^{a_i}\right) \cdot e(g^t, w^{\lceil \log(s) \rceil} z^s h) \\ &= e\left(g, \prod_{i=1}^{n} \left(u_0 \prod_{j=1}^{k} u_j^{m_{i,j}}\right)^{a_i} \cdot (w^{\lceil \log(s) \rceil} z^s h)^t\right) \end{aligned}$$

Therefore, since $e(g, \cdot)$ is bijective (see Lemma 2.2.12), it follows that $\sigma_{\mathsf{Agg},1}$ is of the form

$$\sigma_{\mathsf{Agg},1} = \prod_{i=1}^{n} \left(u_0 \prod_{j=1}^{k} u_j^{m_{i,j}}\right)^{a_i} \cdot (w^{\lceil \log(s) \rceil} z^s h)^t.$$

Since we know sk_x, we can compute the value

$$\sigma_x := \left(u_0 \prod_{j=1}^{k} u_j^{m_{x,j}}\right)^{a_x}.$$

Now, the signature $\sigma'_{\mathsf{Agg}} \coloneqq (\sigma_{\mathsf{Agg},1} \cdot \sigma_x^{-1}, \sigma_{\mathsf{Agg},2}, s)$ is a valid signature for the claim sequence $C \setminus ((\mathsf{pk}_x, m_x))$, since

$$\sigma_{\mathsf{Agg},1} \cdot \sigma_x^{-1} = \prod_{i=1, i\neq x}^{n} \left(u_0 \prod_{j=1}^{k} u_j^{m_{i,j}} \right)^{a_i} \cdot (w^{\lceil \log(s) \rceil} z^s h)^t,$$

which proves that the AGH synchronized aggregate signature scheme is claim-removable.

□

Theorem 5.3.41. *If the Computational Diffie-Hellman assumption holds, then the* AGH *synchronized aggregate signature scheme is* 2DEAGG *secure.*

Proof. The general proof strategy is similar to the proof of the 2DEAGG security of the LOSSW scheme in Theorem 5.3.32. The simulator $\mathcal{B}$ sets up the parameters so that an individual signature for m_1 or m_2 must contain g^{xy}, but does this in such a way that he can still compute a valid aggregate signature, i.e. the aggregate signature does not contain g^{xy}.

Let $\mathcal{C}_{\mathsf{CDH}}$ be the Computational Diffie-Hellman challenger and $\mathcal{A}$ a PPT 2DEAGG attacker on the AGH scheme. We construct a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to solve the Computational Diffie-Hellman problem. See Figure 5.10 for a rough overview of the proof strategy.

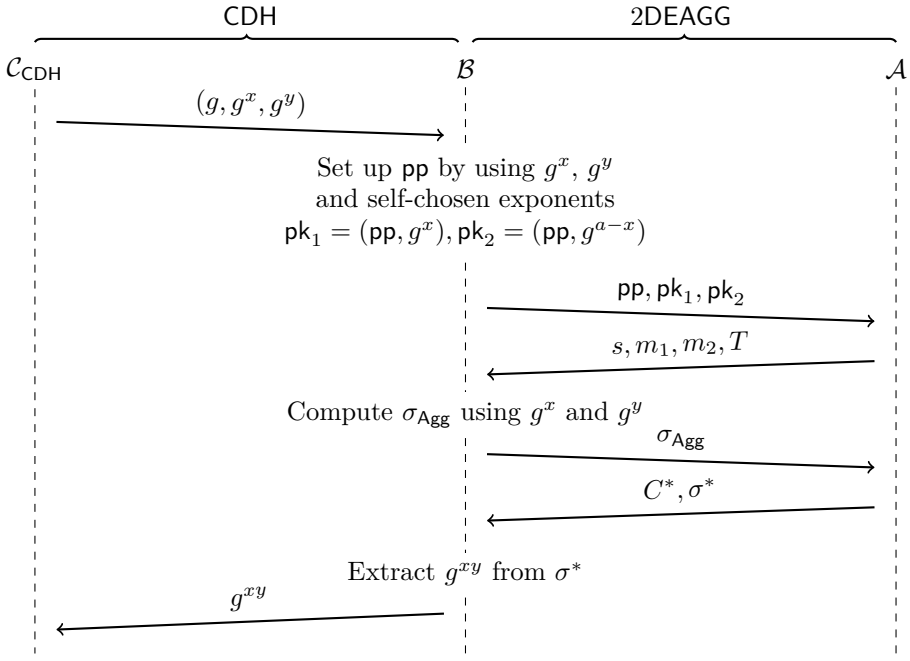

Figure 5.10: Overview of the proof strategy for Theorem 5.3.41.

First, $\mathcal{C}_{\mathsf{CDH}}$ sends a CDH challenge tuple (g, g^x, g^y) to $\mathcal{B}$. Next, $\mathcal{B}$ will not run the $\mathsf{Setup}_{\mathsf{AGH}}$ algorithm, but rather choose the parameters itself in a way that is indistinguishable from an honest execution of $\mathsf{Setup}_{\mathsf{AGH}}$. $\mathcal{B}$ proceeds as follows:

- The group $\mathbb{G}$ is the same that is being used by $\mathcal{C}_{\mathsf{CDH}}$ with prime order p. As its generator, $\mathcal{B}$ chooses g from the CDH challenge tuple.
- ℓ and k are chosen normally, for example $k = 5$ and $\ell = 32$ [AGH10].
- Next $\mathcal{B}$ chooses random values $r_1, \ldots, r_k$ and q_1, q_2, q_3 from $\mathbb{Z}_p$ and sets

$$\begin{aligned} u_0 &\coloneqq g^y, \\ u_i &\coloneqq g^{r_i} \text{ for } 1 \leq i \leq k, \\ w &\coloneqq g^{q_1}, \\ z &\coloneqq g^{q_2}, \\ h &\coloneqq g^{q_3}. \end{aligned}$$

  All of these elements are uniformly distributed (u_0 is randomly distributed, since y is chosen randomly by $\mathcal{C}_{\mathsf{CDH}}$) and are therefore distributed like the values chosen by the $\mathsf{Setup}_{\mathsf{AGH}}$ algorithm.

- $\mathcal{B}$ now sets

$$\mathsf{pp} \coloneqq (\ell, k, p, \mathbb{G}, g, u_0, \ldots, u_k, w, z, h),$$

  which $\mathcal{A}$ cannot distinguish from public parameters output by $\mathsf{Setup}_{\mathsf{AGH}}$.

- For the public keys, $\mathcal{B}$ picks a random $a \leftarrow \mathbb{Z}_p$ and sets

$$\begin{aligned} \mathsf{pk}_1 &\coloneqq (\mathsf{pp}, g^x), \\ \mathsf{pk}_2 &\coloneqq (\mathsf{pp}, g^{a-x}). \end{aligned}$$

  Both keys are random and uniformly distributed, since x is chosen randomly by $\mathcal{C}_{\mathsf{CDH}}$. They are also independent from one another, since g is a generator of $\mathbb{G}$ and a was chosen randomly. Therefore, $\mathcal{A}$ cannot distinguish them from keys created by $\mathsf{Gen}_{\mathsf{AGH}}$.

  Note that by setting the public keys in this way, $\mathcal{B}$ implicitly sets $\mathsf{sk}_1 \coloneqq (\mathsf{pp}, x)$ and $\mathsf{sk}_2 \coloneqq (\mathsf{pp}, a - x)$, which he does not know.

$\mathcal{B}$ now sends $\mathsf{pp}, \mathsf{pk}_1$ and pk_2 to the attacker $\mathcal{A}$, who at some point answers by sending two messages m_1 and m_2, a timestamp s and an order-tree T, for which $\mathcal{B}$ needs to compute an aggregate signature, where m_1 is signed under pk_1 and m_2 under pk_2 using the given time stamp s.

We now show that $\mathcal{B}$ is able to compute such an aggregate signature, although he does not know sk_1 or sk_2 and cannot compute individual signatures for m_1 and m_2.

To see this, suppose that $\sigma_i = (\sigma_{i,1}, g^{t_i}, s)$ is *some* valid AGH signature for $m_i = m_{i,1} \| \ldots \| m_{i,k}$ under pk_i for $i \in \{1, 2\}$. Lemma 5.3.37 implies that

$$\begin{aligned} \sigma_{1,1} &= \left(u_0 \prod_{i=1}^{k} u_i^{m_{1,i}} \right)^x \cdot (w^{\lceil \log(s) \rceil} z^s h)^{t_1} \\ \sigma_{2,1} &= \left(u_0 \prod_{i=1}^{k} u_i^{m_{2,i}} \right)^{a-x} \cdot (w^{\lceil \log(s) \rceil} z^s h)^{t_2} \end{aligned}$$

According to the $\mathsf{Agg}_{\mathsf{AGH}}$ algorithm, an aggregate signature $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg}_1}, \sigma_{\mathsf{Agg}_2}, s)$ for $((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2))$ would therefore be given by

$$\begin{aligned}
\sigma_{\mathsf{Agg},1} &= \sigma_{1,1} \cdot \sigma_{2,1} \\
&= \left(u_0 \prod_{i=1}^{k} u_i^{m_{1,i}}\right)^{x} \cdot (w^{\lceil \log(s) \rceil} z^s h)^{t_1} \cdot \left(u_0 \prod_{i=1}^{k} u_i^{m_{2,i}}\right)^{a-x} \cdot (w^{\lceil \log(s) \rceil} z^s h)^{t_2} \\
&= u_0^{x+a-x} \cdot \left(\prod_{i=1}^{k} (u_i)^{m_{1,i}}\right)^{x} \cdot \left(\prod_{i=1}^{k} (u_i)^{m_{2,i}}\right)^{a-x} \cdot (w^{\lceil \log(s) \rceil} z^s h)^{t_1+t_2} \\
&= g^{ya} \cdot \prod_{i=1}^{k} ((g^x)^{r_i m_{1,i}}) \cdot \prod_{i=1}^{k} \left((g^{a-x})^{r_i m_{2,i}}\right) \cdot (g^{q_1 \lceil \log(s) \rceil + q_2 s + q_3})^{t_1+t_2}.
\end{aligned}$$

and $\sigma_{\mathsf{Agg}} = g^{t_1+t_2}$. Note that this is true for any order-tree T, since $\mathsf{Agg}_{\mathsf{AGH}}$ basically only multiplies group elements in a commutative group, see also the argumentation in Lemma 5.3.37.

Now, since $\mathcal{B}$ knows g^x, g^y, a, $r_1, \ldots, r_k$, m_1, m_2, s, q_1, q_2 and q_3, $\mathcal{B}$ can actually compute a valid aggregate signature for the claims (pk_1, m_1) and (pk_2, m_2) by first choosing a random $t \leftarrow \mathbb{Z}_p$ and then setting

$$\sigma_{\mathsf{Agg}} \coloneqq (g^{ya} \cdot \prod_{i=1}^{k} ((g^x)^{r_i m_{1,i}}) \cdot \prod_{i=1}^{k} \left((g^{a-x})^{r_i m_{2,i}}\right) \cdot (g^{q_1 \lceil \log(s) \rceil + q_2 s + q_3})^{t}, g^t, s).$$

One can also easily verify that this signature fulfills the rules of $\mathsf{Vfy}_{\mathsf{AGH}}$, i.e.

$$\mathsf{Vfy}_{\mathsf{AGH}}(((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2)), \sigma_{\mathsf{Agg}}) = 1.$$

$\mathcal{B}$ now computes such an aggregate signature and sends it to $\mathcal{A}$, who will answer with a claim sequence C^* and corresponding signature $\sigma^* = (\sigma_1^*, \sigma_2^* = g^{t^*}, s^*)$. If $\mathcal{A}$ successfully deaggregated one of the claims, it holds that $\mathsf{Vfy}_{\mathsf{AGH}}(C^*, \sigma^*) = 1$ and $C^* = ((\mathsf{pk}_i, m_i))$ for $i \in \{1, 2\}$.

Case $C^* = ((\mathsf{pk}_1, m_1))$: According to Lemma 5.3.37, we have that

$$\begin{aligned}
\sigma_1^* &= \left(u_0 \prod_{i=1}^{k} u_i^{m_{1,i}}\right)^{x} \cdot (w^{\lceil \log(s^*) \rceil} z^{s^*} h)^{t^*} \\
&= g^{xy} \cdot \prod_{i=1}^{k} ((g^x)^{r_i m_{1,i}}) \cdot (g^{t^*})^{q_1 \lceil \log(s^*) \rceil + q_2 s^* + q_3}
\end{aligned}$$

$\mathcal{B}$ can therefore compute

$$g^{xy} = \sigma_1^* \cdot \left(\prod_{i=1}^{k} ((g^x)^{r_i m_{1,i}}) \cdot (\sigma_2^*)^{q_1 \lceil \log(s^*) \rceil + q_2 s^* + q_3}\right)^{-1}$$

Case $C^* = ((\mathsf{pk}_2, m_2))$: According to Lemma 5.3.37, we have that

$$\begin{aligned}\sigma_1^* &= \left(u_0 \prod_{i=1}^{k} u_i^{m_{2,i}}\right)^{a-x} \cdot \left(w^{\lceil \log(s^*)\rceil} z^{s^*} h\right)^{t^*} \\ &= g^{-xy} \cdot \prod_{i=1}^{k} ((g^{a-x})^{r_i m_{2,i}}) \cdot (g^{t^*})^{q_1 \lceil \log(s^*)\rceil + q_2 s^* + q_3}.\end{aligned}$$

$\mathcal{B}$ can therefore compute

$$g^{xy} = \left(\sigma_1^* \cdot \left(\prod_{i=1}^{k} ((g^{a-x})^{r_i m_{2,i}}) \cdot (\sigma_2^*)^{q_1 \lceil \log(s^*)\rceil + q_2 s^* + q_3}\right)^{-1}\right)^{-1}.$$

In both cases, $\mathcal{B}$ successfully computes g^{xy}, which $\mathcal{B}$ now sends to $\mathcal{C}_{\mathsf{CDH}}$ to win the experiment.

$\mathcal{B}$ simulates the 2DEAGG security experiment perfectly for $\mathcal{A}$. The success probability of $\mathcal{B}$ is greater than or equal to the success probability of $\mathcal{A}$ and their runtimes are roughly the same (the runtime of $\mathcal{B}$ is basically the run time of $\mathcal{A}$ plus the time it takes to set up the parameters, to compute the public keys and to extract g^{xy} at the end, which is all polynomial). It follows that the success probability of $\mathcal{A}$ must be negligible, since we assumed that the CDH assumption holds. Therefore the 2DEAGG security of the AGH synchronized aggregate signature scheme follows.

□

Theorem 5.3.42. *The* AGH *synchronized aggregate signature scheme is* nDEAGG *secure for all* $n \geq 2$ *polynomial in the security parameter* κ *under the Computational Diffie-Hellman assumption.*

Proof. The theorem follows directly from Theorem 5.2.21, Theorem 5.3.41 and the lemmas 5.3.38, 5.3.39 and 5.3.40. □

Unfortunately, just like most previous schemes, the AGH synchronized aggregate signature scheme is *not* nDEAGG$^+$ secure.

Theorem 5.3.43. *The* AGH *synchronized aggregate signature scheme is* not nDEAGG$^+$ *secure for any* n.

Proof. Consider the following PPT attacker $\mathcal{A}$ on the nDEAGG$^+$ experiment. First, $\mathcal{A}$ receives public parameter $\mathsf{pp} = (\ell, k, p, \mathbb{G}, g, u_0, \ldots, u_k, w, z, h)$ and n public keys $\mathsf{pk}_i = (\mathsf{pp}, g^{a_i})$ from the challenger. $\mathcal{A}$ chooses n random messages $m_i = m_{i,1} \| \ldots \| m_{i,k}$ consisting of k blocks of ℓ bits, and sends them and a fitting order-tree (for example the order-tree describing sequential aggregation) to the challenger, so that m_i gets signed under pk_i. The challenger answers with a valid signature $\sigma_{\mathsf{Agg}} = (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2}, s)$ for the claim sequence $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$, where s is the timestamp and $\sigma_{\mathsf{Agg},2} = g^t$ for some exponent $t \in \mathbb{Z}_p$.

$\mathcal{A}$ will now manipulate the signature σ_{Agg} so that it becomes valid for a new claim sequence $C' = (c_1', ((\mathsf{pk}_2, m_2), \ldots, (\mathsf{pk}_n, m_n)))$ for a new claim $c_1' = (\mathsf{pk}', m_1) \neq (\mathsf{pk}_1, m_1)$. To do this, $\mathcal{A}$ sets

$$\mathsf{pk}' = (\mathsf{pp}, g^{a_1} \cdot g) = (\mathsf{pp}, g^{a_1+1}).$$

Clearly, we have $c_1' \neq (\mathsf{pk}_1, m_1)$ and therefore $C \neq C'$. Since σ_{Agg} is valid and was computed by the challenger by correctly applying the $\mathsf{Sign}_{\mathsf{AGH}}$ and $\mathsf{Agg}_{\mathsf{AGH}}$ algorithms, it holds that

$$\sigma_{\mathsf{Agg},1} = \left(u_0 \prod_{i=1}^{k} u_i^{m_{1,i}}\right)^{a_1} \cdot \ldots \cdot \left(u_0 \prod_{i=1}^{k} u_i^{m_{n,i}}\right)^{a_n} (w^{\lceil \log(s) \rceil} z^s h)^t.$$

Next, $\mathcal{A}$ multiplies $\sigma_{\mathsf{Agg},1}$ with the value $\sigma_1' = \left(u_0 \prod_{i=1}^{k} u_i^{m_{1,i}}\right)$ to compute

$$\sigma_{\mathsf{Agg},1} \cdot \sigma_1' = \left(u_0 \prod_{i=1}^{k} u_i^{m_{1,i}}\right)^{a_1+1} \cdot \prod_{j=2}^{n} \left(u_0 \prod_{i=1}^{k} u_i^{m_{j,i}}\right)^{a_j} \cdot (w^{\lceil \log(s) \rceil} z^s h)^t$$

and sets $\sigma' = (\sigma_{\mathsf{Agg},1} \cdot \sigma_1', \sigma_{\mathsf{Agg},2}, s)$, which is a valid signature for the claim sequence C'. $\mathcal{A}$ outputs C' and σ' and wins the experiment, since the claim (pk_1, m_1) no longer appears in the claim sequence. The success probability of $\mathcal{A}$ is equal to 1 and its runtime is polynomial. It follows that the AGH scheme is *not* $n\mathsf{DEAGG}^+$ secure.

□

The problem and attack strategy is essentially the same as for the BGLS scheme: the attacker changes the public key of one of the claims, but leaves the messages as they were before. Note that, also like in the attack on the BGLS scheme, the attacker could set $\mathsf{pk}' = (\mathsf{pp}, g^{a_1} \cdot g^r)$ and $\sigma_1' = \left(u_0 \prod_{i=1}^{k} u_i^{m_{1,i}}\right)^r$ for a random exponent $r \in \mathbb{Z}_p \setminus \{0\}$ to completely hide the connection to pk_1.

Theorem 5.3.44. *The* AGH *synchronized aggregate signature scheme is* not ADEAGG *secure.*

Proof. The theorem follows from Theorem 5.2.34 and Theorem 5.3.43 directly above. It also follows from the fact that aggregation can be inverted. An attacker can easily output a weakly signable claim sequence together with a fitting signature by inverting the group elements of other signatures.

For example, if $\sigma_1 = (\sigma_{1,1}, \sigma_{1,2}, s)$ is a signature for $C_1 = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2), (\mathsf{pk}_3, m_3))$ and $\sigma_2 = (\sigma_{2,1}, \sigma_{2,2}, s)$ a signature for $C_2 = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_3, m_3))$, then

$$\sigma' = \left(\frac{\sigma_{1,1}}{\sigma_{2,1}}, \frac{\sigma_{1,2}}{\sigma_{2,2}}, s\right)$$

is a valid signature for the claim sequence $C' = ((\mathsf{pk}_2, m_2))$. If the attacker only queries signatures for C_1 and C_2, then C' is weakly signable and the attacker can use it to win the experiment with success probability 1. □

5.3.8 The AGHRO Synchronized Aggregate Signature Scheme

Ahn, Green, and Hohenberger [AGH10] also give a more efficient variant of their scheme, which however is only secure in the random oracle model:

Definition 5.3.45 (AGHRO Synchronized Aggregate Signature Scheme)**.** *The* AGHRO *synchronized aggregate signature scheme* $\Sigma_{\mathsf{AGHRO}} = (\mathsf{Setup}_{\mathsf{AGHRO}}, \mathsf{Gen}_{\mathsf{AGHRO}}, \mathsf{Sign}_{\mathsf{AGHRO}}, \mathsf{Agg}_{\mathsf{AGHRO}}, \mathsf{Vfy}_{\mathsf{AGHRO}})$ *consists of five PPT algorithms as follows:*

$\mathsf{Setup}_{\mathsf{AGHRO}}(1^\kappa)$**.** *Select a bilinear group* $\mathbb{G}$ *of prime order* $p > 2^\kappa$ *and a pairing* $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$*. Let* $H : \{0,1\}^* \to \mathbb{Z}_p$ *be a hash function treated as a random oracle. Choose random elements* $g, u, v, w, z, h \in \mathbb{G}$*. Output the public parameters as*

$$\mathsf{pp} = (\mathbb{G}, \mathbb{G}_T, H, g, u, v, w, z, h).$$

$\mathsf{Gen}_{\mathsf{AGHRO}}(1^\kappa, \mathsf{pp})$**.** *Pick a random* $a \leftarrow \mathbb{Z}_p$*. Set* $\mathsf{pk} := (\mathsf{pp}, g^a)$ *and* $\mathsf{sk} := (\mathsf{pp}, a)$*, initialize* s_{prev} *to zero and return* $(\mathsf{pk}, \mathsf{sk})$*.*

$\mathsf{Sign}_{\mathsf{AGHRO}}(\mathsf{sk}, m, s)$**.** *Abort, if* $s = s_{\mathsf{prev}}$ *or* $s \geq 2^\kappa$*. Otherwise set* $s_{\mathsf{prev}} := s$ *(it is assumed that the input* s *is the current time period as output by* $\mathsf{clock}()$*). Select a random* $t \in \mathbb{Z}_p$*, compute*

$$\sigma_1 := \left(vu^{H(m)}\right)^a \cdot (w^{\lceil \log(s) \rceil} z^s h)^t,$$
$$\sigma_2 := g^t,$$

and return $\sigma = (\sigma_1, \sigma_2, s)$ *as the signature.*

$\mathsf{Agg}_{\mathsf{AGHRO}}(C_1, C_2, \sigma_1, \sigma_2)$**.** *Parse* σ_i *as* $(\sigma_{i,1}, \sigma_{i,2}, s_i)$ *and check that* $s_1 = s_2$ *and* $\mathsf{Vfy}_{\mathsf{AGHRO}}(C_i, \sigma_i) = 1$ *for* $i \in \{1, 2\}$*. If any of these check fails, output* $\perp$*. Otherwise compute*

$$\sigma_{\mathsf{Agg},1} := \sigma_{1,1} \cdot \sigma_{2,1},$$
$$\sigma_{\mathsf{Agg},2} := \sigma_{1,2} \cdot \sigma_{2,2},$$

and output $\sigma_{\mathsf{Agg}} := (\sigma_{\mathsf{Agg},1}, \sigma_{\mathsf{Agg},2}, s_1)$*.*

$\mathsf{Vfy}_{\mathsf{AGHRO}}(C, \sigma)$**.** *Parse* $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$ *and* $\sigma = (\sigma_1, \sigma_2, s)$*. Check that* $0 < s < 2^\kappa$*. If this is not the case, abort and output 0. Extract* g^{a_i} *from each* pk_i *and check that*

$$e(g, \sigma_1) = e(\prod_{i=1}^{n} g^{a_i}, v) \cdot e(\prod_{i=1}^{n} g^{a_i H(m_i)}, u) \cdot e(\sigma_2, w^{\lceil \log(s) \rceil} z^s h).$$

Output 1 *if this check is successful, otherwise output* 0*.*

This variant exhibits the same properties as the AGH scheme in the standard model. We give the respective theorems below, the proofs are largely analogous to the proofs of the theorems above.

Theorem 5.3.46. *The* AGHRO *synchronized aggregate signature scheme is* SyncAS-EUF-CMA *secure in the random oracle model, if the computational Diffie-Hellman assumption holds and H is modeled as a random oracle.*

Proof. A proof of this theorem can be found in Appendix B of [AGH10]. □

Lemma 5.3.47. *The* AGHRO *synchronized aggregate signature scheme is extendable, claim-removable and order-independent.*

Proof. The proof of this lemma is analogous to the proofs of the Lemmas 5.3.38, 5.3.39 and 5.3.40. □

Theorem 5.3.48. *If the Computational Diffie-Hellman assumption holds, then the* AGHRO *synchronized aggregate signature scheme is* 2DEAGG *secure in the random oracle model.*

Proof. This proof is largely analogous to the proof of Theorem 5.3.41. We therefore omit detailed discussions of some formalities to keep the proof short. Let $\mathcal{C}_{\mathsf{CDH}}$ be the Computational Diffie-Hellman challenger and $\mathcal{A}$ a PPT attacker on the AGHRO scheme. We construct a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to solve the Computational Diffie-Hellman problem.

First, $\mathcal{C}_{\mathsf{CDH}}$ passes on a CDH challenge tuple (g, g^x, g^y) to $\mathcal{B}$. Next, $\mathcal{B}$ will choose the parameters of the scheme in a way that is indistinguishable from an honest execution of $\mathsf{Setup}_{\mathsf{AGHRO}}$:

- The group $\mathbb{G}$ is the same that is being used by $\mathcal{C}_{\mathsf{CDH}}$ with prime order p. As its generator, $\mathcal{B}$ chooses g from the CDH challenge tuple.
- $\mathcal{B}$ chooses random values r and q_1, q_2, q_3 from $\mathbb{Z}_p$ and sets
$$\begin{aligned} u &\coloneqq g^r, \\ v &\coloneqq g^y, \\ w &\coloneqq g^{q_1}, \\ z &\coloneqq g^{q_2}, \\ h &\coloneqq g^{q_3}. \end{aligned}$$
All of these elements are uniformly distributed.
- $\mathcal{B}$ now sets
$$\mathsf{pp} \coloneqq (\mathbb{G}, \mathbb{G}_T, H, g, u, v, w, z, h),$$
where H is the random oracle. $\mathcal{A}$ cannot distinguish these parameters from honestly generated parameters output by $\mathsf{Setup}_{\mathsf{AGHRO}}$.
- For the public keys, $\mathcal{B}$ picks a random $a \leftarrow \mathbb{Z}_p$ and sets
$$\begin{aligned} \mathsf{pk}_1 &\coloneqq (\mathsf{pp}, g^x), \\ \mathsf{pk}_2 &\coloneqq (\mathsf{pp}, g^{a-x}). \end{aligned}$$
Both keys are random and uniformly distributed, since x is chosen randomly by $\mathcal{C}_{\mathsf{CDH}}$. They are also independent from one another, since g is a generator and a was chosen randomly. Therefore, $\mathcal{A}$ cannot distinguish them from keys created by $\mathsf{Gen}_{\mathsf{AGHRO}}$.

  Note that by setting the public keys in this way, $\mathcal{B}$ implicitly sets $\mathsf{sk}_1 \coloneqq (\mathsf{pp}, x)$ and $\mathsf{sk}_2 \coloneqq (\mathsf{pp}, a - x)$.

$\mathcal{B}$ now sends $\mathsf{pp}, \mathsf{pk}_1$ and pk_2 to the attacker $\mathcal{A}$. For the rest of the experiment, all random oracle queries to H will be answered honestly by $\mathcal{B}$, i.e. if $\mathcal{A}$ sends a random oracle query for $H(m)$, $\mathcal{B}$ first checks whether this value was already set. If yes, $\mathcal{B}$ answers with this stored value, if not, $\mathcal{B}$ chooses a new random value from $\mathbb{Z}_p$, sends it as the answer and stores it.

At some point, $\mathcal{A}$ answers by sending two messages m_1 and m_2, a timestamp s and an order-tree T for which $\mathcal{B}$ needs to compute an aggregate signature, where m_1 is signed under pk_1 and m_2 under pk_2 using the given time stamp s.

Similar to Lemma 5.3.37 one can prove that individual signatures of the AGHRO scheme are always of the form $\sigma = (\sigma_1, \sigma_2, s)$ where

$$\begin{aligned}\sigma_1 &\coloneqq \left(vu^{H(m)}\right)^a \cdot (w^{\lceil \log(s) \rceil} z^s h)^t,\\ \sigma_2 &\coloneqq g^t,\end{aligned}$$

for some $t \in \mathbb{Z}_p$ and where a is the exponent of the corresponding secret key. Similar to the proof of Theorem 5.3.41, it follows that a valid signature for $C = ((\mathsf{pk}_1, m_1), (\mathsf{pk}_2, m_2))$ is given by

$$\sigma_{\mathsf{Agg}} = (g^{ya} \cdot (g^x)^{H(m_1)r} \cdot (g^{a-x})^{H(m_2)r} \cdot (g^{q_1 \lceil \log(s) \rceil + q_2 s + q_3})^t, g^t, s).$$

for any order-tree T. It is also easy to see that this signature fulfills the rules of $\mathsf{Vfy}_{\mathsf{AGHRO}}$, i.e. we have $\mathsf{Vfy}_{\mathsf{AGHRO}}(C, \sigma_{\mathsf{Agg}}) = 1$. Since $\mathcal{B}$ knows the values g^y, a, r, m_1, m_2, q_1, q_2, q_3, s and can choose t randomly, $\mathcal{B}$ can compute such a signature and send it to $\mathcal{A}$, who will answer with C^* and $\sigma^* = (\sigma_1^*, \sigma_2^* = g^{t^*}, s^*)$. If $\mathcal{A}$ successfully deaggregated one of the claims, it holds that $\mathsf{Vfy}_{\mathsf{AGHRO}}(C^*, \sigma^*) = 1$ and $C^* = ((\mathsf{pk}_i, m_i))$ for $i \in \{1, 2\}$.

Case $C^* = ((\mathsf{pk}_1, m_1))$**:** We have that

$$\begin{aligned}\sigma_1^* &= v^x \cdot u^{H(m)x} \cdot (w^{\lceil \log(s^*) \rceil} z^{s^*} h)^{t^*}\\ &= g^{xy} \cdot (g^x)^{H(m_1)r} \cdot (g^{t^*})^{q_1 \lceil \log(s^*) \rceil + q_2 s^* + q_3}.\end{aligned}$$

$\mathcal{B}$ can therefore compute

$$g^{xy} = \sigma_1^* \cdot \left((g^x)^{H(m_1)r} (\sigma_2^*)^{q_1 \lceil \log(s^*) \rceil + q_2 s^* + q_3}\right)^{-1}.$$

Case $C^* = ((\mathsf{pk}_2, m_2))$**:** We have that

$$\begin{aligned}\sigma_1^* &= v^{a-x} \cdot u^{H(m_2)a} \cdot u^{-H(m_2)x} \cdot (w^{\lceil \log(s^*) \rceil} z^{s^*} h)^{t^*}\\ &= g^{-xy} \cdot g^{-ya} \cdot g^{H(m_2)ar} \cdot (g^x)^{-H(m_2)r} \cdot (g^{t^*})^{q_1 \lceil \log(s^*) \rceil + q_2 s^* + q_3}.\end{aligned}$$

$\mathcal{B}$ can therefore compute

$$g^{xy} = \left(\sigma_1^* \cdot \left(g^{-ya} \cdot g^{H(m_2)ar} \cdot (g^x)^{-H(m_2)r} \cdot (\sigma_2^*)^{q_1 \lceil \log(s^*) \rceil + q_2 s^* + q_3}\right)^{-1}\right)^{-1}.$$

In both cases, $\mathcal{B}$ can use σ^* to compute g^{xy}, which $\mathcal{B}$ now sends to $\mathcal{C}_{\mathsf{CDH}}$ to win the challenge.

$\mathcal{B}$ simulates the 2DEAGG security experiment perfectly for $\mathcal{A}$. The success probability of $\mathcal{B}$ is greater than or equal to the success probability of $\mathcal{A}$ and their runtimes are roughly the same (the runtime of $\mathcal{B}$ is basically the runtime of $\mathcal{A}$ plus the time it takes to set up the parameters and to extract g^{xy} at the end, which is all polynomial). It follows that the success probability of $\mathcal{A}$ must be negligible, since we assumed that the CDH assumption holds. Therefore the 2DEAGG security of the AGHRO synchronized aggregate signature scheme follows.

□

Remark. Note that the proof above never takes explicit advantage of the fact that H is a random oracle. However, since the scheme is only proven to be unforgeable in the random oracle model, we also only state the theorem in this model.

Theorem 5.3.49. *If the Computational Diffie-Hellman assumption holds, then the* AGHRO *synchronized aggregate signature scheme is* nDEAGG *secure in the random oracle model for all* $n \geq 2$ *polynomial in the security parameter* κ*.*

Proof. The theorem follows directly from Theorem 5.2.21, Theorem 5.3.48 and Lemma 5.3.47. □

Theorem 5.3.50. *The* AGHRO *synchronized aggregate signature scheme is neither* nDEAGG$^+$ *secure for any* n *nor* ADEAGG *secure.*

Proof. The proof of this theorem is analogous to the proofs of Theorem 5.3.43 and Theorem 5.3.44. □

Since the AGHRO scheme is also only secure in the random oracle model, an interesting question is whether modifications similar to the ones by Bellare, Namprempre, and Neven [BNN07] (see Section 5.3.2) and Saxena, Misra, and Dhar [SMD14] (see Section 5.3.3) applied to the BGLS scheme could be used to make the scheme nDEAGG$^+$ or ADEAGG secure. Instead of signing $H(m)$, Bellare et al. sign $H(\mathsf{pk}\|m)$ and Saxena et al. sign $H(m\|r\|\mathsf{pk})$ for a random value $r \leftarrow \{0,1\}^\kappa$, that is then also stored in the signature. However, we leave this question open, especially since the modification of Saxena et al. has a drastic negative effect on the signature size (signatures now grow linearly in the number of claims) and aggregation no longer compresses the signatures.

5.3.9 Overview of the Schemes

To conclude, we give an overview over the deaggregation security of the known aggregate signature schemes that we examined in this thesis in Table 5.1 below. The column "Security" states the *strongest* form of deaggregation security provided and the used computational assumption. Note that this overview is not complete and there exist several more schemes which deaggregation security needs to be researched, like for example [GR06; LLY13b; LLY13a; HW18].

Scheme	Type	Model	Security	Comment
BGLS [Bon+03]	Full	RO	nDEAGG (CDH)	
BNN [BNN07]	Full	RO	nDEAGG$^+$ (CDH)	The scheme is called $\mathcal{AS}$-3 in [BNN07].
SMD [SMD14]	Full	RO	ADEAGG (CDH)	Signature size grows *linearly* in the number of claims.
LMRS [Lys+04]	Seq.	RO	None	
NEV [Nev08]	Seq.	RO	None	
LOSSW [Lu+06]	Seq.	Stand.	nDEAGG (CDH)	
AGH [AGH10]	Sync.	Stand.	nDEAGG (CDH)	
AGHRO [AGH10]	Sync.	RO	nDEAGG (CDH)	More efficient variant of AGH, but only proven secure in the random oracle model.

Table 5.1: An overview of the deaggregation security of the schemes discussed in this thesis. "Full" stands for fully flexible aggregation, "Seq." for sequential aggregation and "Sync." for synchronized aggregation. "RO" stands for random oracle model, "Stand." for standard model and "CDH" for Computational Diffie-Hellman Assumption.

5.4 Deaggregation Security & Fault-Tolerance

Since both fault-tolerance and deaggregation security are important concepts for many applications of aggregate signature schemes, it seems like the final goal would be to construct an efficient fully flexible scheme that is *both* fault-tolerant and deaggregation secure. Unfortunately, as we show in this section, it is not possible to achieve this goal.

In fact, deaggregation *insecurity* is virtually a "built-in feature" of fault-tolerant schemes, since deaggregating a signature or modifying a claim basically inserts one fault. To see this, let σ_{Agg} be a valid aggregate signature for a claim sequence $C = (c_1, \ldots, c_n)$ that was computed by aggregating individual signatures σ_i for c_i, i.e. "$\sigma_{\mathsf{Agg}} = \prod_{i=1}^{n} \sigma_i$". Now, simply removing the claim c_n from C will mean that the already aggregated signature σ_n is no longer regular, i.e. this removal introduces one fault. Since the scheme is fault-tolerant, it will

therefore only output the claims c_1 to c_{n-1}, so the claim c_n was successfully deaggregated. We prove this formally in the following theorem.

Again, we only discuss the case of fully flexible aggregate signature schemes in detail, the results however also transfer to sequential and synchronized schemes.

Remark. To be formally precise, we would first need to restate the deaggregation security definitions to consider fault-tolerance and list verification. This could be done by demanding that the set of claims output by Vfy contains *all* non-empty claims of the claim sequence sent by the attacker, as well as the normal requirement that at least one claim must have been deaggregated (either by allowing additional claims for $n\mathsf{DEAGG}^+$ security or not for $n\mathsf{DEAGG}$ security), i.e. at least one of the old claims is not part of the output of Vfy after the attack. If the scheme needs some order among the claims (like in our construction from Chapter 4), then the challenger will also use the order described by the order-tree for this purpose. Since these changes are rather straightforward, we forgo formally defining these experiments again.

Theorem 5.4.1. *Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ be an aggregate signature scheme with list verification. If Σ is d-fault-tolerant for any $d \in \mathbb{N}$, then Σ is* not *$n\mathsf{DEAGG}$ secure for any $n \leq d$.*

Proof. Consider the following PPT attacker $\mathcal{A}$ on the $n\mathsf{DEAGG}$ experiment. First, $\mathcal{A}$ receives a list of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ from the challenger $\mathcal{C}_{n\mathsf{DEAGG}}$. $\mathcal{A}$ then chooses n messages $m_1, \ldots, m_n$ at random and sends $(m_1, \ldots, m_n), T$ to $\mathcal{C}_{n\mathsf{DEAGG}}$, where T is a fitting order-tree for the n messages. Next, $\mathcal{A}$ receives a valid aggregate signature τ_{Agg} for $C = ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n))$.

Now, let τ_i be the individual signatures computed by the challenger for the claim (pk_i, m_i), which was used to compute τ_{Agg}. Observe that $\tau_i \neq \lambda$, since $(\mathsf{pk}_i, m_i) \neq \perp$. Then $M = \{(c_1, \tau_1), \ldots, (c_n, \tau_n)\}$ is the multiset of claim-signature pairs associated with C and τ_{Agg}. Since all signatures τ_i and τ_{Agg} were computed correctly, M contains 0 errors and we have $M_{\text{reg}} = M$ and $M_{\text{irreg}} = \emptyset$.

The attacker now defines a new claim sequence $C' := ((\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_{n-1}, m_{n-1}), \perp)$, does *not* modify τ_{Agg} at all and simply outputs C', τ_{Agg}.

The associated multiset of claim-signature pairs now is $M' = \{(m_1, \tau_1), \ldots, (m_{n-1}, \tau_{n-1}), (\perp, \tau_n)\}$, since *all* n signatures τ_i were used to compute τ_{Agg}. Since $\tau_n \neq \lambda$, we have that $M'_{\text{reg}} = (((\mathsf{pk}_1, m_1), \tau_1), \ldots, ((\mathsf{pk}_{n-1}, m_{n-1}), \tau_{n-1})))$ and $M_{\text{irreg}} = ((\perp, \tau_n))$, i.e. M' contains 1 error.

Since Σ is d-fault-tolerant for some $d \geq 1$, we have that $M'_{\text{reg}} \subseteq \mathsf{Vfy}(C', \tau_{\mathsf{Agg}})$. Furthermore, since (pk_n, m_n) is not in $\mathsf{elem}(C')$, we also have that $(\mathsf{pk}_n, m_n) \notin \mathsf{Vfy}(C', \tau_{\mathsf{Agg}})$ by definition, since the Vfy algorithm of a fault-tolerant scheme never outputs claims that are not parts of its input (see Definition 4.3.1).

It follows that $\mathcal{A}$ has successfully deaggregated the claim (pk_n, m_n) and therefore wins the $n\mathsf{DEAGG}$ experiment. $\mathcal{A}$ is a PPT algorithm and has success probability 1, which proves the theorem.

□

Corollary 5.4.2. *Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vfy})$ be an aggregate signature scheme with list verification. If Σ is d-fault-tolerant for any $d \in \mathbb{N}$, then Σ is* neither *$n\mathsf{DEAGG}^+$ secure for any $n \leq d$ nor ADEAGG secure.*

Proof. The corollary follows directly from Theorem 5.2.9, Theorem 5.2.34 and Theorem 5.4.1. □

The question now is if we can find some other *meaningful* definition of deaggregation security that *can* be achieved by fault-tolerant schemes.

First, note that the proof can easily be changed so that $\mathcal{A}$ deaggregates a whole set of claims (as long as the number of removed claims is less than or equal to d). Therefore the scheme remains insecure even if we would change the winning condition in the nDEAGG security experiment to require that more than one claim needs to be deaggregated.

Only if we'd require that more than d claims need to be deaggregated could the scheme potentially be nDEAGG secure. However, we are of the opinion that a change like this is not constructive and would remove the definition further from real-life applications, especially since the goal is to provide d-fault-tolerance for a d that is as large as possible.

Another way could be to *not* interpret such removals of claims from the sequence as an error, but this would only be sound for schemes where such removals do not influence the validity of other claims. Therefore, while it is not obvious how such a scheme could work or be constructed, it seems questionable if this approach is reasonable. For example, if we look at schemes based on algebraic groups, like the BGLS scheme, we can see that such "claim removals" *without* also removing the corresponding signature parts from the aggregate will always invalidate the whole aggregate, i.e. they clearly should be interpreted as an error. This is similar to aggregating a random element of the signature space, without knowing or including its corresponding claim in the verification, which should render any aggregate signature of any non-fault-tolerant scheme invalid.

No matter how we try to redefine nDEAGG security, nDEAGG$^+$ and especially ADEAGG security is generally unachievable. If the attacker $\mathcal{A}$ does not simply remove the claim (pk_1, m_1), but instead exchanges it with a random claim (pk', m'), $\mathcal{A}$ would always win the respective security experiments.

Since a modification of a claim like this should *clearly* be interpreted as an error in the sense of fault-tolerance, it seems like there is no good way to combine fault-tolerance and deaggregation security. If both fault-tolerance and deaggregation security is needed, one therefore needs to carefully decide which feature should be directly provided by the aggregate signature scheme and which feature needs to be achieved by additional security measures. For example, while deaggregation security would be advantageous for secure logging, our construction from Section 4.7 shows that a similar protection can be achieved by combining a standard digital signature scheme with a fault-tolerant sequential aggregate signature scheme. The digital signature scheme is used to compute a signature on the length of the log, which, together with unforgeability of both schemes, ensures that no entry can be removed. However, this approach only works since the way the log changes over time is very clearly defined and is only directly modified by one party (i.e. the server) that can create and store the additional secret key and signature. If an aggregate signature scheme should be used to protect a more complex database, then such a simple approach might not work. Instead, it might be favorable to use a deaggregation secure scheme and additional strong back-up mechanisms to provide "fault-tolerance".

Remark. Theorem 5.4.1 of course also holds for our fault-tolerant black-box construction using a standard aggregate signature scheme and a cover-free family presented in Chapter 4. Even if the underlying aggregate signature scheme might be deaggregation secure, the resulting fault-tolerant scheme will not be.

Chapter 6

Conclusion and Prospects

In this thesis, we discussed fault-tolerance and deaggregation security of aggregate signature schemes. Aggregate signature schemes are a type of digital signature that offer an additional aggregation algorithm that can be used to compress several signatures over multiple claims into a single small signature.

All aggregate signature schemes presented so far have the shared problem that signatures become invalid as soon as one invalid signature is aggregated. In contrast, signatures of a fault-tolerant scheme can withstand the aggregation of invalid signatures, so that they can still be used to validate the correctly signed messages after invalid signatures were aggregated.

We precisely defined the concept of fault-tolerance for all three major types of aggregation (fully flexible, synchronized and sequential aggregation) and gave the *first* construction of a fault-tolerant aggregate signature scheme. The construction uses a cover-free family and a standard aggregate signature scheme as building blocks. It is a black-box transformation that can be used to turn *any* aggregate signature scheme into a fault-tolerant scheme and has a tight security proof. We show that fault-tolerant schemes cannot offer perfect compression (i.e. aggregate signatures need to grow in size and cannot be of the same size as individual signatures), if at least a constant number of faults is to be tolerated. We also show that signatures of our scheme only grow logarithmically in the number of claims, which is optimal.

Our construction has two restrictions: First, claims are identified with subsets of the cover-free family. As a result, claims need to be assigned to a fixed position in the claim sequence, which means that aggregation is slightly restricted. It is only possible to aggregate signatures that have no overlap in the already occupied positions of their claim sequences. However, this restriction is only minor, since signatures can still be aggregated in any order and in many applications signers can easily agree on such fixed positions for their claims. Second, and more importantly, the number of claims that can be aggregated is bounded by the size of the universe of the cover-free family. We also showed an unbounded construction, however its signatures unfortunately grow linearly in the number of claims. As a practical application, we showed how fault-tolerant sequential aggregate signatures can be used to construct a robust and secure logging scheme.

The second concept that we discussed, namely deaggregation security, provides security against the unwanted removal of claims from an aggregate signature. Given an aggregate signature of a set of messages, an attacker might be able to

use it to compute a new signature for a subset of these messages by removing or deaggregating individual signatures. We introduced several security definitions capturing a basic protection against deaggregation attacks, namely nDEAGG and nDEAGG$^+$ security. Both definitions provide security against attackers that can ask for one aggregate signature of n claims and then need to deaggregate at least one claim from it. For nDEAGG, the attacker is not allowed to aggregate additional claims for his attack, while nDEAGG$^+$ allows this. These definitions are comparatively weak, but suffice for interesting applications, like constructing verifably encrypted signature schemes.

We examined the relationship of these definitions and showed that there exists a formal hierarchy, i.e. different levels for different n do not necessarily imply one another. However, we also showed that for aggregate signature schemes that exhibit three natural properties, called extendability, claim-removability and order-independence (that might also be of independent interest), nDEAGG security implies n'DEAGG security for $n' > n$ (the same holds for nDEAGG$^+$ security).

Furthermore, we showed that unforgeability does not imply any form of deaggregation security. We also discussed and generalized the security definition of Saxena, Misra, and Dhar [SMD14] (which we called ADEAGG security) that implies a high level of security against deaggregation attacks and showed how it relates to the other definitions. Also, using our hierarchy of security definitions, we precisely classified the deaggregation security provided by several known aggregate signature schemes by giving new security proofs and attacks. Since most schemes were not designed with deaggregation security in mind, almost all of them only provide nDEAGG or nDEAGG$^+$ security. The only scheme to provide ADEAGG security is the scheme of [SMD14], which has the drawbacks that signatures grow linearly in the number of aggregated claims and that it is only proven secure in the random oracle model.

Finally, we also discussed the connection between fault-tolerance and deaggregation security. We showed that no fault-tolerant scheme can offer any form of deaggregation security, which implies that for applications of aggregate signature schemes, a trade-off between both properties needs to be made. It needs to be carefully decided which property should be provided by the aggregate signature scheme and if other measures need to be taken (i.e. using additional cryptographic primitives or security mechanisms) to achieve a protection of the application similar to that provided by the other property.

In the following, we briefly discuss some research questions that arise from our work. An interesting open problem is to find fault-tolerant constructions with a very small or even no "gap" between the number d of faults that can be tolerated and the number n of aggregated claims. The question if there exist d-fault-tolerant schemes were d is not fixed a priori, but can grow in proportion to the number of aggregated claims, also remains open. Note that there is a trivial solution to achieve both goals: The aggregation algorithm simply stores all individual signatures in a list. While this construction is very inefficient and it is questionable if it should even be interpreted as an "aggregate" signature scheme, it hints at the existence of such schemes. Furthermore, the theoretically best number of how many faults can be tolerated by schemes that actually *do* compress their input is unknown.

Another interesting research direction would be to search for other building blocks than cover-free families that can be used to construct fault-tolerant schemes and to investigate if they can achieve better bounds then our construction.

Since signatures of fault-tolerant schemes necessarily need to grow in size if at least a constant number d of faults is to be tolerated, it would be interesting to research whether a trade-off between constant signature size and d can be made, for example by investigating if schemes can be constructed that offer constant signature size, but where d slowly degrades in the number of aggregated claims.

In our fault-tolerant construction, signers need to assign positions to their claims, i.e. aggregation is slightly restricted and not fully flexible. This is unproblematic for many applications, especially since we do not impose an order of aggregation, but it would still be interesting to construct a fault-tolerant scheme that offers fully flexible aggregation. Furthermore, only a limited number of signatures can be aggregated. While we also construct an unbounded scheme, we only achieve a constant compression ratio. Idalino and Moura [IM18] already improved our construction by using *nested* cover-free families and achieve better compression ratios. However, there is still room for improvement, since they also do not achieve the optimal compression ratio (i.e. signatures that only grow logarithmically), if more than two faults are to be tolerated.

Moreover, we assume that aggregation itself is always done correctly. While it is impossible to give guarantees for all possible errors that can happen during aggregation (a faulty aggregator could output a random string), it would be interesting to investigate certain classes of errors that can occur in practice due to common software or hardware errors.

Another interesting research direction would be to investigate whether schemes can be constructed where partially invalid aggregate signatures could also somehow efficiently and securely be "repaired", such that the signature again becomes valid for all of its claims.

For deaggregation security, most known (and efficient) aggregate signature schemes unfortunately only provide basic nDEAGG security and are vulnerable to stronger attacks. The important problem of constructing an efficient aggregate signature scheme (of any type) in the standard model that provides a stronger form of protection against deaggregation attacks remains open for now. In particular, there exists no efficient aggregate signature scheme with short signatures that is also ADEAGG secure, neither in the standard model, nor in the random oracle model.

Since a large number of fully flexible, sequential and synchronized aggregate signature schemes have been proposed, we could only examine the deaggregation security of a limited number of schemes in this thesis. It would therefore be interesting to exactly classify the security of the remaining schemes as well.

Furthermore, while we showed that nDEAGG security implies n'DEAGG security for $n' > n$ for schemes that are extendable, claim-removable and order-independent and showed that order-independence is a necessary property, it remains open whether extendability and claim-removability are also necessary for this implication to hold. The same question remains open for extendability for $n\mathsf{DEAGG}^+$ security (here, claim-removability is not necessary).

Bibliography

[AB09] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

[AGH10] J. H. Ahn, M. Green, and S. Hohenberger. "Synchronized aggregate signatures: new definitions, constructions and applications." In: *ACM CCS 10*. Ed. by E. Al-Shaer, A. D. Keromytis, and V. Shmatikov. ACM Press, Oct. 2010, pp. 473–484.

[And97] R. Anderson. *Invited Lecture. 4th ACM Computer and Communications Security*. 1997.

[Bad+16] C. Bader, T. Jager, Y. Li, and S. Schäge. "On the Impossibility of Tight Cryptographic Reductions." In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. 2016, pp. 273–304. DOI: 10.1007/978-3-662-49896-5_10. URL: https://doi.org/10.1007/978-3-662-49896-5_10.

[Bar+04] B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. "Universally Composable Protocols with Relaxed Set-Up Assumptions." In: *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*. 2004, pp. 186–195. DOI: 10.1109/FOCS.2004.71. URL: https://doi.org/10.1109/FOCS.2004.71.

[BGN05] D. Boneh, E. Goh, and K. Nissim. "Evaluating 2-DNF Formulas on Ciphertexts." In: *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*. 2005, pp. 325–341. DOI: 10.1007/978-3-540-30576-7_18. URL: https://doi.org/10.1007/978-3-540-30576-7_18.

[BGR12] K. Brogle, S. Goldberg, and L. Reyzin. "Sequential Aggregate Signatures with Lazy Verification from Trapdoor Permutations - (Extended Abstract)." In: *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*. 2012, pp. 644–662. DOI: 10.1007/978-3-642-34961-4_39. URL: https://doi.org/10.1007/978-3-642-34961-4_39.

[BGR14] K. Brogle, S. Goldberg, and L. Reyzin. "Sequential aggregate signatures with lazy verification from trapdoor permutations." In: *Inf. Comput.* 239 (2014), pp. 356–376. DOI: 10.1016/j.ic.2014.07.001. URL: https://doi.org/10.1016/j.ic.2014.07.001.

[BJ10] A. Bagherzandi and S. Jarecki. "Identity-Based Aggregate and Multi-Signature Schemes Based on RSA." In: *PKC 2010*. Ed. by P. Q. Nguyen and D. Pointcheval. Vol. 6056. LNCS. Springer, May 2010, pp. 480–498.

[BLS04] D. Boneh, B. Lynn, and H. Shacham. "Short Signatures from the Weil Pairing." In: *J. Cryptology* 17.4 (2004), pp. 297–319. DOI: 10.1007/s00145-004-0314-9. URL: https://doi.org/10.1007/s00145-004-0314-9.

[BM99a] M. Bellare and S. K. Miner. "A Forward-Secure Digital Signature Scheme." In: 1999, pp. 431–448. DOI: 10.1007/3-540-48405-1_28.

[BM99b] S. Blake-Wilson and A. Menezes. "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol." In: *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings*. 1999, pp. 154–170. DOI: 10.1007/3-540-49162-7_12. URL: https://doi.org/10.1007/3-540-49162-7_12.

[BMP16] R. E. Bansarkhani, M. S. E. Mohamed, and A. Petzoldt. "MQSAS - A Multivariate Sequential Aggregate Signature Scheme." In: *Information Security - 19th International Conference, ISC 2016, Honolulu, HI, USA, September 3-6, 2016, Proceedings*. 2016, pp. 426–439. DOI: 10.1007/978-3-319-45871-7_25. URL: https://doi.org/10.1007/978-3-319-45871-7_25.

[BNN07] M. Bellare, C. Namprempre, and G. Neven. "Unrestricted Aggregate Signatures." In: *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*. 2007, pp. 411–422. DOI: 10.1007/978-3-540-73420-8_37. URL: https://doi.org/10.1007/978-3-540-73420-8_37.

[Bol+07] A. Boldyreva, C. Gentry, A. O'Neill, and D. H. Yum. "Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing." In: *ACM CCS 07*. Ed. by P. Ning, S. D. C. di Vimercati, and P. F. Syverson. ACM Press, Oct. 2007, pp. 276–285.

[Bon+03] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps." In: *EUROCRYPT 2003*. Ed. by E. Biham. Vol. 2656. LNCS. Springer, May 2003, pp. 416–432.

[Bon+07] D. Boneh, R. Canetti, S. Halevi, and J. Katz. "Chosen-Ciphertext Security from Identity-Based Encryption." In: *SIAM J. Comput.* 36.5 (2007), pp. 1301–1328. DOI: 10.1137/S009753970544713X. URL: https://doi.org/10.1137/S009753970544713X.

[Boy+06] X. Boyen, H. Shacham, E. Shen, and B. Waters. "Forward-secure signatures with untrusted update." In: 2006, pp. 191–200. DOI: 10.1145/1180405.1180430.

[BR93] M. Bellare and P. Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols." In: *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.* 1993, pp. 62–73. DOI: 10.1145/168588.168596. URL: https://doi.org/10.1145/168588.168596.

[BR94] M. Bellare and P. Rogaway. "Optimal Asymmetric Encryption." In: *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings.* 1994, pp. 92–111. DOI: 10.1007/BFb0053428. URL: https://doi.org/10.1007/BFb0053428.

[BR96] M. Bellare and P. Rogaway. "The Exact Security of Digital Signatures - How to Sign with RSA and Rabin." In: *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding.* 1996, pp. 399–416. DOI: 10.1007/3-540-68339-9_34. URL: https://doi.org/10.1007/3-540-68339-9_34.

[BY97] M. Bellare and B. Yee. *Forward Integrity for Secure Audit Logs.* Tech. rep. Computer Science and Engineering Department, University of California at San Diego, 1997.

[CGH04] R. Canetti, O. Goldreich, and S. Halevi. "The random oracle methodology, revisited." In: *J. ACM* 51.4 (2004), pp. 557–594. DOI: 10.1145/1008731.1008734. URL: https://doi.org/10.1145/1008731.1008734.

[Che+06] X. Cheng, J. Liu, L. Guo, and X. Wang. "Identity-based multisignature and aggregate signature schemes from m-torsion groups." In: *Journal of Electronics (China)* 23.4 (2006), pp. 569–573. DOI: 10.1007/s11767-004-0178-z. URL: https://doi.org/10.1007/s11767-004-0178-z.

[CL04] J. Camenisch and A. Lysyanskaya. "Signature Schemes and Anonymous Credentials from Bilinear Maps." In: *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings.* 2004, pp. 56–72. DOI: 10.1007/978-3-540-28628-8_4. URL: https://doi.org/10.1007/978-3-540-28628-8_4.

[CL19] E. C. Crites and A. Lysyanskaya. "Delegatable Anonymous Credentials from Mercurial Signatures." In: *Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings.* 2019, pp. 535–555. DOI: 10.1007/978-3-030-12612-4_27. URL: https://doi.org/10.1007/978-3-030-12612-4_27.

[CLW05] X. Cheng, J. Liu, and X. Wang. "Identity-Based Aggregate and Verifiably Encrypted Signatures from Bilinear Pairing." In: *Computational Science and Its Applications - ICCSA 2005, International Conference, Singapore, May 9-12, 2005, Proceedings, Part IV*. 2005, pp. 1046–1054. DOI: 10.1007/11424925_109. URL: https://doi.org/10.1007/11424925_109.

[CN03] J. Coron and D. Naccache. "Boneh et al.'s k-Element Aggregate Extraction Assumption Is Equivalent to the Diffie-Hellman Assumption." In: *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*. 2003, pp. 392–397. DOI: 10.1007/978-3-540-40061-5_25. URL: https://doi.org/10.1007/978-3-540-40061-5_25.

[Coh+05] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, eds. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman and Hall/CRC, 2005. DOI: 10.1201/9781420034981. URL: https://doi.org/10.1201/9781420034981.

[Cra+07] R. Cramer, G. Hanaoka, D. Hofheinz, H. Imai, E. Kiltz, R. Pass, a. shelat, and V. Vaikuntanathan. "Bounded CCA2-Secure Encryption." In: *ASIACRYPT 2007*. Ed. by K. Kurosawa. Vol. 4833. LNCS. Springer, Dec. 2007, pp. 502–518.

[CS98] R. Cramer and V. Shoup. "A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack." In: *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*. 1998, pp. 13–25. DOI: 10.1007/BFb0055717. URL: https://doi.org/10.1007/BFb0055717.

[DH76] W. Diffie and M. E. Hellman. "New directions in Cryptography." In: *IEEE Trans. Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638. URL: https://doi.org/10.1109/TIT.1976.1055638.

[DMR00a] A. G. Dyachkov, A. J. Macula, and V. V. Rykov. "New applications and results of superimposed code theory arising from the potentialities of molecular biology." In: *Numbers, Information and Complexity*. Ed. by I. Althöfer, N. Cai, G. Dueck, L. Khachatrian, M. Pinsker, A. Sárközy, I. Wegener, and Z. Zhang. Springer, 2000, pp. 265–282.

[DMR00b] A. G. Dyachkov, A. J. Macula, and V. V. Rykov. "New constructions of superimposed codes." In: *IEEE Transactions on Information Theory* 46.1 (2000), pp. 284–290. DOI: 10.1109/18.817530.

[Dou+09] B. Dou, H. Zhang, C. Xu, and M. Han. "Identity-Based Sequential Aggregate Signature from RSA." In: *Fourth ChinaGrid Annual Conference, ChinaGrid 2009, Yantai, Shandong, China, 21-22 August, 2009*. 2009, pp. 123–127. DOI: 10.1109/ChinaGrid.2009.20. URL: https://doi.org/10.1109/ChinaGrid.2009.20.

[EGM96] S. Even, O. Goldreich, and S. Micali. "On-Line/Off-Line Digital Signatures." In: *J. Cryptology* 9.1 (1996), pp. 35–67. DOI: 10.1007/BF02254791. URL: https://doi.org/10.1007/BF02254791.

[FLS12] M. Fischlin, A. Lehmann, and D. Schröder. "History-Free Sequential Aggregate Signatures." In: *Security and Cryptography for Networks - 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings*. 2012, pp. 113–130. DOI: 10.1007/978-3-642-32928-9_7. URL: https://doi.org/10.1007/978-3-642-32928-9_7.

[GGH12] S. Garg, C. Gentry, and S. Halevi. "Candidate Multilinear Maps from Ideal Lattices and Applications." In: *IACR Cryptology ePrint Archive* 2012 (2012), p. 610. URL: http://eprint.iacr.org/2012/610.

[GMR88] S. Goldwasser, S. Micali, and R. L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks." In: *SIAM J. Comput.* 17.2 (1988), pp. 281–308. DOI: 10.1137/0217017. URL: https://doi.org/10.1137/0217017.

[Gol01] O. Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[Gol04] O. Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511721656. URL: http://www.wisdom.weizmann.ac.il/%5C%7Eoded/foc-vol2.html.

[GPS08] S. D. Galbraith, K. G. Paterson, and N. P. Smart. "Pairings for cryptographers." In: *Discrete Applied Mathematics* 156.16 (2008), pp. 3113–3121. DOI: 10.1016/j.dam.2007.12.010. URL: https://doi.org/10.1016/j.dam.2007.12.010.

[GR06] C. Gentry and Z. Ramzan. "Identity-Based Aggregate Signatures." In: *PKC 2006*. Ed. by M. Yung, Y. Dodis, A. Kiayias, and T. Malkin. Vol. 3958. LNCS. Springer, Apr. 2006, pp. 257–273.

[Grö16] R. Gröll. "Deaggregation von aggregierbaren Signaturen." Diploma Thesis. Karlsruher Institut für Technologie (KIT), 2016.

[Gui13] A. Guillevic. "Comparing the Pairing Efficiency over Composite-Order and Prime-Order Elliptic Curves." In: *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*. 2013, pp. 357–372. DOI: 10.1007/978-3-642-38980-1_22. URL: https://doi.org/10.1007/978-3-642-38980-1_22.

[Har+16] G. Hartung, B. Kaidel, A. Koch, J. Koch, and A. Rupp. "Fault-Tolerant Aggregate Signatures." In: *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*. 2016, pp. 331–356. DOI: 10.1007/978-3-662-49384-7_13. URL: https://doi.org/10.1007/978-3-662-49384-7_13.

[Har+17a] G. Hartung, B. Kaidel, A. Koch, J. Koch, and D. Hartmann. "Practical and Robust Secure Logging from Fault-Tolerant Sequential Aggregate Signatures." In: *Provable Security - 11th International Conference, ProvSec 2017, Xi'an, China, October 23-25, 2017, Proceedings*. 2017, pp. 87–106. DOI: 10.1007/978-3-319-68637-0_6. URL: https://doi.org/10.1007/978-3-319-68637-0_6.

[Har+17b] G. Hartung, B. Kaidel, A. Koch, J. Koch, and D. Hartmann. "Practical and Robust Secure Logging from Fault-Tolerant Sequential Aggregate Signatures." In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 949. URL: http://eprint.iacr.org/2017/949. This is the full version of [Har+17a].

[Har16] G. Hartung. "Secure Audit Logs with Verifiable Excerpts." In: *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*. 2016, pp. 183–199. DOI: 10.1007/978-3-319-29485-8_11. URL: https://doi.org/10.1007/978-3-319-29485-8_11.

[Har17] G. Hartung. "Attacks on Secure Logging Schemes." In: *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*. 2017, pp. 268–284. DOI: 10.1007/978-3-319-70972-7_14. URL: https://doi.org/10.1007/978-3-319-70972-7_14.

[Har20] G. Hartung. "Advanced Cryptographic Techniques for Protecting Log Data." PhD thesis. Karlsruher Institut für Technologie (KIT), 2020.

[Her06] J. Herranz. "Deterministic Identity-Based Signatures for Partial Aggregation." In: *Comput. J.* 49.3 (2006), pp. 322–330. DOI: 10.1093/comjnl/bxh153.

[HJ12] D. Hofheinz and T. Jager. "Tightly Secure Signatures and Public-Key Encryption." In: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. 2012, pp. 590–607. DOI: 10.1007/978-3-642-32009-5_35. URL: https://doi.org/10.1007/978-3-642-32009-5_35.

[HJK11] D. Hofheinz, T. Jager, and E. Kiltz. "Short Signatures from Weaker Assumptions." In: *ASIACRYPT 2011*. Ed. by D. H. Lee and X. Wang. Vol. 7073. LNCS. Springer, Dec. 2011, pp. 647–666.

[HKW15] S. Hohenberger, V. Koppula, and B. Waters. "Universal Signature Aggregators." In: *EUROCRYPT 2015, Part II*. Ed. by E. Oswald and M. Fischlin. Vol. 9057. LNCS. Springer, 2015, pp. 3–34. DOI: 10.1007/978-3-662-46803-6_1.

[HLY09] J. Y. Hwang, D. H. Lee, and M. Yung. "Universal forgery of the identity-based sequential aggregate signature scheme." In: *Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009, Sydney, Australia, March 10-12, 2009*. 2009, pp. 157–160. DOI: 10.1145/1533057.1533080. URL: https://doi.org/10.1145/1533057.1533080.

[Hol06] J. E. Holt. "Logcrypt: forward security and public verification for secure audit logs." In: *The proceedings of the Fourth Australasian Symposium on Grid Computing and e-Research (AusGrid 2006) and the Fourth Australasian Information Security Workshop (Network Security) (AISW 2006), Hobart, Tasmania, Australia, January 2006*. 2006, pp. 203–211. URL: https://dl.acm.org/citation.cfm?id=1151852.

[HSW13] S. Hohenberger, A. Sahai, and B. Waters. "Full Domain Hash from (Leveled) Multilinear Maps and Identity-Based Aggregate Signatures." In: *CRYPTO 2013, Part I*. Ed. by R. Canetti and J. A. Garay. Vol. 8042. LNCS. Springer, Aug. 2013, pp. 494–512. DOI: 10.1007/978-3-642-40041-4_27.

[HW09] S. Hohenberger and B. Waters. "Realizing Hash-and-Sign Signatures under Standard Assumptions." In: *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*. 2009, pp. 333–350. DOI: 10.1007/978-3-642-01001-9_19. URL: https://doi.org/10.1007/978-3-642-01001-9_19.

[HW18] S. Hohenberger and B. Waters. "Synchronized Aggregate Signatures from the RSA Assumption." In: *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*. 2018, pp. 197–229. DOI: 10.1007/978-3-319-78375-8_7. URL: https://doi.org/10.1007/978-3-319-78375-8_7.

[Ida+15] T. B. Idalino, L. Moura, R. F. Custódio, and D. Panario. "Locating modifications in signed data for partial data integrity." In: *Inf. Process. Lett.* 115.10 (2015), pp. 731–737. DOI: 10.1016/j.ipl.2015.02.014. URL: https://doi.org/10.1016/j.ipl.2015.02.014.

[Ida15] T. B. Idalino. "Using combinatorial group testing to solve integrity issues." Universidade Federal de Santa Catarina, Brazil, 2015. URL: https://repositorio.ufsc.br/handle/123456789/169646?show=full. Last accessed February 5, 2020.

[IM18] T. B. Idalino and L. Moura. "Efficient Unbounded Fault-Tolerant Aggregate Signatures Using Nested Cover-Free Families." In: *Combinatorial Algorithms - 29th International Workshop, IWOCA 2018, Singapore, July 16-19, 2018, Proceedings*. 2018, pp. 52–64. DOI: 10.1007/978-3-319-94667-2_5. URL: https://doi.org/10.1007/978-3-319-94667-2_5.

[Kat10] J. Katz. *Digital Signatures*. Springer, 2010. DOI: 10.1007/978-0-387-27712-7. URL: https://doi.org/10.1007/978-0-387-27712-7.

[Ken+00] S. T. Kent, C. Lynn, J. Mikkelson, and K. Seo. "Secure Border Gateway Protocol (S-BGP) - Real World Performance and Deployment Issues." In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA*. 2000. URL: http://www.isoc.org/isoc/conferences/ndss/2000/proceedings/045.pdf.

[KL14] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[KM15] N. Koblitz and A. J. Menezes. "The random oracle model: a twenty-year retrospective." In: *Des. Codes Cryptogr.* 77.2-3 (2015), pp. 587–610. DOI: 10.1007/s10623-015-0094-2. URL: https://doi.org/10.1007/s10623-015-0094-2.

[Koc19] J. Koch. "Improvements and New Constructions of Digital Signatures." PhD thesis. Karlsruher Institut für Technologie (KIT), 2019. DOI: 10.5445/IR/1000097524.

[KRS99] R. Kumar, S. Rajagopalan, and A. Sahai. "Coding Constructions for Blacklisting Problems without Computational Assumptions." In: *CRYPTO '99*. Ed. by M. J. Wiener. Vol. 1666. LNCS. Springer, Aug. 1999, pp. 609–623.

[KS64] W. H. Kautz and R. C. Singleton. "Nonrandom binary superimposed codes." In: *IEEE Transactions on Information Theory* 10.4 (1964), pp. 363–377. DOI: 10.1109/TIT.1964.1053689.

[Lam79] L. Lamport. *Constructing Digital Signatures from a One Way Function*. Tech. rep. CSL-98. This paper was published by IEEE in the Proceedings of HICSS-43 in January, 2010. Oct. 1979. URL: https://www.microsoft.com/en-us/research/publication/constructing-digital-signatures-one-way-function/. Last accessed February 5, 2020.

[LLY13a] K. Lee, D. H. Lee, and M. Yung. "Sequential Aggregate Signatures Made Shorter." In: *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*. 2013, pp. 202–217. DOI: 10.1007/978-3-642-38980-1_13. URL: https://doi.org/10.1007/978-3-642-38980-1_13.

[LLY13b] K. Lee, D. H. Lee, and M. Yung. "Sequential Aggregate Signatures with Short Public Keys: Design, Analysis and Implementation Studies." In: *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*. 2013, pp. 423–442. DOI: 10.1007/978-3-642-36362-7_26. URL: https://doi.org/10.1007/978-3-642-36362-7_26.

[Lu+06] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. "Sequential Aggregate Signatures and Multisignatures Without Random Oracles." In: *EUROCRYPT 2006*. Ed. by S. Vaudenay. Vol. 4004. LNCS. Springer, May 2006, pp. 465–485.

[LVW06] P. Li, G. Van Rees, and R. Wei. "Constructions of 2-cover-free families and related separating hash families." In: *Journal of Combinatorial Designs* 14.6 (2006), pp. 423–440.

[LVY01] V. Lebedev, P. Vilenkin, and S. Yekhanin. "Cover-free families and superimposed codes: Constructions, bounds, and applications to cryptography and group testing." In: *In IEEE International Symposium on Information Theory*. 2001.

[Lys+04] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. "Sequential Aggregate Signatures from Trapdoor Permutations." In: *EUROCRYPT 2004*. Ed. by C. Cachin and J. Camenisch. Vol. 3027. LNCS. Springer, May 2004, pp. 74–90.

[Lys+99] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. "Pseudonym Systems." In: *Selected Areas in Cryptography, 6th Annual International Workshop, SAC'99, Kingston, Ontario, Canada, August 9-10, 1999, Proceedings*. 1999, pp. 184–199. DOI: 10.1007/3-540-46513-8_14. URL: https://doi.org/10.1007/3-540-46513-8_14.

[Lys02] A. Lysyanskaya. "Signature schemes and applications to cryptographic protocol design." PhD thesis. Massachusetts Institute of Technology, Cambridge, MA, USA, 2002. URL: http://hdl.handle.net/1721.1/29271.

[Ma08] D. Ma. "Practical forward secure sequential aggregate signatures." In: *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008, Tokyo, Japan, March 18-20, 2008*. 2008, pp. 341–352. DOI: 10.1145/1368310.1368361. URL: https://doi.org/10.1145/1368310.1368361.

[Mer89] R. C. Merkle. "A Certified Digital Signature." In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. 1989, pp. 218–238. DOI: 10.1007/0-387-34805-0_21. URL: https://doi.org/10.1007/0-387-34805-0_21.

[MNT04] E. Mykletun, M. Narasimha, and G. Tsudik. "Signature Bouquets: Immutability for Aggregated/Condensed Signatures." In: *Computer Security - ESORICS 2004, 9th European Symposium on Research Computer Security, Sophia Antipolis, France, September 13-15, 2004, Proceedings*. 2004, pp. 160–176. DOI: 10.1007/978-3-540-30108-0_10. URL: https://doi.org/10.1007/978-3-540-30108-0_10.

[MOV93] A. Menezes, T. Okamoto, and S. A. Vanstone. "Reducing elliptic curve logarithms to logarithms in a finite field." In: *IEEE Trans. Information Theory* 39.5 (1993), pp. 1639–1646. DOI: 10.1109/18.259647. URL: https://doi.org/10.1109/18.259647.

[MS04] A. Menezes and N. P. Smart. "Security of Signature Schemes in a Multi-User Setting." In: *Des. Codes Cryptogr.* 33.3 (2004), pp. 261–274. DOI: 10.1023/B:DESI.0000036250.18062.3f. URL: https://doi.org/10.1023/B:DESI.0000036250.18062.3f.

[MT07] D. Ma and G. Tsudik. "Extended Abstract: Forward-Secure Sequential Aggregate Authentication." In: *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*. 2007, pp. 86–91. DOI: 10.1109/SP.2007.18. URL: https://doi.org/10.1109/SP.2007.18.

[MT08] D. Ma and G. Tsudik. "A New Approach to Secure Logging." In: *Data and Applications Security XXII, 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Proceedings*. Ed. by V. Atluri. Vol. 5094. Lecture Notes in Computer Science. Springer, 2008, pp. 48–63. DOI: 10.1007/978-3-540-70567-3_4.

[MT09] D. Ma and G. Tsudik. "A new approach to secure logging." In: *TOS* 5.1 (2009). DOI: 10.1145/1502777.1502779.

[Nak09] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: http://www.bitcoin.org/bitcoin.pdf. Last accessed February 5, 2020.

[Nev08] G. Neven. "Efficient Sequential Aggregate Signed Data." In: *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*. 2008, pp. 52–69. DOI: 10.1007/978-3-540-78967-3_4. URL: https://doi.org/10.1007/978-3-540-78967-3_4.

[Nie02] J. B. Nielsen. "Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-Committing Encryption Case." In: *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*. 2002, pp. 111–126. DOI: 10.1007/3-540-45708-9_8. URL: https://doi.org/10.1007/3-540-45708-9_8.

[NY90] M. Naor and M. Yung. "Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks." In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*. 1990, pp. 427–437. DOI: 10.1145/100216.100273. URL: https://doi.org/10.1145/100216.100273.

[Rab79] M. O. Rabin. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization*. Tech. rep. Cambridge, MA, USA, 1979.

[Res18] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: https://rfc-editor.org/rfc/rfc8446.txt. Last accessed February 5, 2020.

[Rom90] J. Rompel. "One-Way Functions are Necessary and Sufficient for Secure Signatures." In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*. 1990, pp. 387–394. DOI: 10.1145/100216.100269. URL: https://doi.org/10.1145/100216.100269.

[RS13] M. Rückert and D. Schröder. "Aggregate and Verifiably Encrypted Signatures from Multilinear Maps Without Random Oracles." In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 20. URL: http://eprint.iacr.org/2013/020.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." In: *Commun. ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342. URL: http://doi.acm.org/10.1145/359340.359342.

[Sch11] D. Schröder. "How to Aggregate the CL Signature Scheme." In: *ESORICS 2011, Proceedings*. Ed. by V. Atluri and C. Díaz. Vol. 6879. LNCS. Springer, 2011, pp. 298–314. DOI: 10.1007/978-3-642-23822-2_17.

[Sel+12] S. S. D. Selvi, S. S. Vivek, J. Shriram, and C. P. Rangan. "Identity based partial aggregate signature scheme without pairing." In: *2012 35th IEEE Sarnoff Symposium, Newark, NJ, USA, May 21-22, 2012*. 2012, pp. 1–6. DOI: 10.1109/SARNOF.2012.6222731. URL: https://doi.org/10.1109/SARNOF.2012.6222731.

[Sha84] A. Shamir. "Identity-Based Cryptosystems and Signature Schemes." In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*. 1984, pp. 47–53. DOI: 10.1007/3-540-39568-7_5. URL: https://doi.org/10.1007/3-540-39568-7_5.

[SMD14] A. Saxena, J. Misra, and A. Dhar. "Increasing Anonymity in Bitcoin." In: *Financial Cryptography and Data Security - FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers*. 2014, pp. 122–139. DOI: 10.1007/978-3-662-44774-1_9. URL: https://doi.org/10.1007/978-3-662-44774-1_9.

[SW99] R. Safavi-Naini and H. Wang. "Multireceiver Authentication Codes: Models, Bounds, Constructions, and Extensions." In: *Information and Computation* 151.1-2 (1999), pp. 148–172. DOI: 10.1006/inco.1998.2769.

[TLW13] J. Tsai, N. Lo, and T. Wu. "New Identity-Based Sequential Aggregate Signature Scheme from RSA." In: *International Symposium on Biometrics and Security Technologies, ISBAST 2013, 2-5 July, 2013, Chengdu, Sichuan, China*. 2013. URL: http://ieeexplore.ieee.org/document/6597680/.

[TS06] D. Tonien and R. Safavi-Naini. "An Efficient Single-Key Pirates Tracing Scheme Using Cover-Free Families." In: *ACNS 06*. Ed. by J. Zhou, M. Yung, and F. Bao. Vol. 3989. LNCS. Springer, June 2006, pp. 82–97.

[Wat05] B. Waters. "Efficient Identity-Based Encryption Without Random Oracles." In: *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*. 2005, pp. 114–127. DOI: 10.1007/11426639_7. URL: https://doi.org/10.1007/11426639_7.

[XZF05] J. Xu, Z. Zhang, and D. Feng. "ID-Based Aggregate Signatures from Bilinear Pairings." In: *Cryptology and Network Security, 4th International Conference, CANS 2005, Xiamen, China, December 14-16, 2005, Proceedings*. 2005, pp. 110–119. DOI: 10.1007/11599371_10. URL: https://doi.org/10.1007/11599371_10.

Author's Publications

[Har+16] G. Hartung, B. Kaidel, A. Koch, J. Koch, and A. Rupp. "Fault-Tolerant Aggregate Signatures." In: *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I.* 2016, pp. 331–356. DOI: 10.1007/978-3-662-49384-7_13. URL: https://doi.org/10.1007/978-3-662-49384-7_13.

[Har+17a] G. Hartung, B. Kaidel, A. Koch, J. Koch, and D. Hartmann. "Practical and Robust Secure Logging from Fault-Tolerant Sequential Aggregate Signatures." In: *Provable Security - 11th International Conference, ProvSec 2017, Xi'an, China, October 23-25, 2017, Proceedings*. 2017, pp. 87–106. DOI: 10.1007/978-3-319-68637-0_6. URL: https://doi.org/10.1007/978-3-319-68637-0_6.

[Har+17b] G. Hartung, B. Kaidel, A. Koch, J. Koch, and D. Hartmann. "Practical and Robust Secure Logging from Fault-Tolerant Sequential Aggregate Signatures." In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 949. URL: http://eprint.iacr.org/2017/949. This is the full version of [Har+17a].

Copyright

Supervised Student Theses

[Agr14] T. Agrikola. "Angriffe auf neue Sicherheitsforderungen für Chamäleon-Signaturen." Bachelor Thesis. Karlsruher Institut für Technologie (KIT), 2014.

[Did16] C. Didong. "Sicherheitstransformationen für aggregierbare Signaturen." Bachelor Thesis. Karlsruher Institut für Technologie (KIT), 2016.

[Grö16] R. Gröll. "Deaggregation von aggregierbaren Signaturen." Diploma Thesis. Karlsruher Institut für Technologie (KIT), 2016.