Malware-Resistant Protocols for Real-World Systems

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Timon Hackenjos

Tag der mündlichen Prüfung: 4. Juli 2023

1. Referent: PD. Dr.-Ing. Ingmar Baumgart

2. Referent: Prof. Dr. Jörn Müller-Quade

Abstract

Cryptographic protocols are widely used to protect real-world systems from attacks. Paying for goods in a shop, withdrawing money or browsing the Web; all these activities are backed by cryptographic protocols. However, in recent years a potent threat became apparent. Malware is increasingly used in attacks to bypass existing security mechanisms. Many cryptographic protocols that are used in real-world systems today have been found to be susceptible to malware attacks [56, 93, 129, 130]. One reason for this is that most of these protocols were designed with respect to the Dolev-Yao attack model that assumes an attacker to control the network between computer systems but not the systems themselves. Furthermore, most real-world protocols do not provide a formal proof of security and thus lack a precise definition of the security goals the designers tried to achieve. This work tackles the design of cryptographic protocols that are resilient to malware attacks, applicable to real-world systems, and provably secure.

In this regard, we investigate three real-world use cases: electronic payment, web authentication, and data aggregation. We analyze the security of existing protocols and confirm results from prior work that most protocols are not resilient to malware. Furthermore, we provide guidelines for the design of malware-resistant protocols and propose such protocols. In addition, we formalize security notions for malware-resistance and use a formal proof of security to verify the security guarantees of our protocols.

In this work we show that designing malware-resistant protocols for real-world systems is possible. We present a new security notion for electronic payment and web authentication, called *one-out-of-two security*, that does not require a single device to be trusted and ensures that a protocol stays secure as long as one of two devices is not compromised. Furthermore, we propose L-Pay, a cryptographic protocol for paying at the point of sale (POS) or withdrawing money at an automated teller machine (ATM) satisfying one-out-of-two security, FIDO2 With Two Displays (FIDO2D) a cryptographic protocol to secure transactions in the Web with one-out-of-two security and Secure Aggregation Grouped by Multiple Attributes (SAGMA), a cryptographic protocol for secure data aggregation in encrypted databases.

In this work, we take important steps towards the use of malware-resistant protocols in real-world systems. Our guidelines and protocols can serve as templates to design new cryptographic protocols and improve security in further use cases.

Zusammenfassung

Kryptografische Protokolle sind weit verbreitet und werden oft dafür eingesetzt Systeme in der echten Welt vor Angriffen zu schützen. Beim Bezahlen von Waren in einem Geschäft, dem Geldabheben am Automaten und Surfen im Internet laufen im Hintergrund kryptografische Protokolle ab. Jedoch zeichnet sich in den letzten Jahren eine mächtige Bedrohung ab. Bei immer mehr Angriffen kommt Malware zum Einsatz und wird genutzt, um bestehende Sicherheitsmechanismen zu umgehen.

Viele heutzutage in der realen Welt eingesetzte kryptografische Protokolle stellten sich in der Vergangenheit als anfällig für Malware-Angriffe heraus [56, 93, 129, 130]. Ein Grund dafür ist, dass viele dieser Protokolle auf Basis des Dolev-Yao Angreifermodells entworfen wurden, das davon ausgeht, dass der Angreifer das Netzwerk zwischen mehreren Computern kontrolliert, jedoch nicht die Systeme selbst. Des Weiteren besitzen die meisten kryptografischen Protokolle, die in der Realität eingesetzt werden, keinen formalen Sicherheitsbeweis und somit auch keine präzise Definition der Sicherheitsziele, auf die die Entwickler des Protokolls abzielten. Diese Arbeit behandelt den Entwurf kryptografischer Protokolle, die Schutz vor Malware bieten, in der Realität anwendbar und beweisbar sicher sind.

Hierfür untersuchen wir drei realitätsnahe Anwendungsfälle: elektronische Bezahlverfahren, die Authentifizierung im Web und die Aggregation von Daten. Wir analysieren die Sicherheit existierender Protokolle und bestätigen die Ergebnisse früherer Arbeiten, dass die meisten Protokolle anfällig für Malware-Angriffe sind. Des Weiteren stellen wir Richtlinien für den Entwurf kryptografischer Protokolle vor, die Schutz vor Malware-Angriffen bieten und schlagen auf dieser Basis neue Protokolle mit dieser Eigenschaft vor. Zusätzlich formalisieren wir Sicherheitsbegriffe für den Schutz gegen Malware und verwenden formale Sicherheitsbeweise, um diese Sicherheitsgarantien nachzuweisen.

In dieser Arbeit zeigen wir, dass es möglich ist kryptografische Protokolle zu entwerfen, die einen Schutz vor Malware bieten und in der echten Welt anwendbar sind. Wir führen einen neuen Sicherheitsbegriff für elektronische Bezahlverfahren und zur Authentifizierung im Web ein, den wir *one-out-of-two security* nennen. Dieser Sicherheitsbegriff erfordert nicht, dass ein einzelnes System als vertrauenswürdig angenommen wird und stellt sicher, dass ein Protokoll Schutz bietet, sofern eines von zwei Geräten nicht kompromittiert ist. Des Weiteren führen wir L-Pay ein, ein kryptografisches Protokoll zum Bezahlen in Geschäften oder Geldabheben am Geldautomaten, das *one-out-of-two security* erfüllt, FIDO2D ein kryptografisches Protokoll zur Absicherung von Transaktionen im Web mit *one-out-of-two security* und SAGMA, ein kryptografisches Protokoll für die sichere Aggregation von Daten in verschlüsselten Datenbanken.

In dieser Arbeit unternehmen wir wichtige Schritte in Richtung des Einsatzes kryptografischer Protokolle, die Schutz vor Malware bieten, in der echten Welt. Unsere Richtlinien und Protokolle dienen als Vorlagen, um neue kryptografische Protokolle zu entwerfen und die Sicherheit in weiteren Anwendungsfällen zu erhöhen.

Danksagung

Ich möchte allen Personen danken, die mich bei der Erstellung dieser Arbeit unterstützt haben. Zuallererst gilt mein Dank meinem Erstgutachter Ingmar Baumgart, der die Arbeit von Anfang an begleitet hat. Ebenfalls danken möchte ich den ehemaligen Kollegen von SAP, die während der Masterarbeit mein Interesse am wissenschaftlichen Arbeiten weckten. Insbesondere möchte ich meinen damaligen Betreuer und Koautor Florian Hahn nennen, der mich über die gesamte Dauer der Arbeit unterstützt hat. Ebenfalls danken möchte ich Andreas Fischer für den wertvollen Austausch während unserer gemeinsamen Zeit bei SAP und am FZI.

Des Weiteren möchte ich meinen ehemaligen Kollegen vom FZI für die vielen Diskussionen und die gute gemeinsame Zeit danken. Ein besonderer Dank gilt Jochen Rill für die jahrelange Ermutigung und Unterstützung, sowie Julian Herr für die vielen Stunden, in denen wir uns gemeinsam mit verschiedensten Forschungsfragen auseinandergesetzt haben. Ebenfalls danken möchte ich meinem ehemaligen Hiwi und Koautor Benedikt Wagner der stets ausgezeichnete Arbeit lieferte und mich bei der Umsetzung von FIDO2D unterstützte.

Ich möchte allen danken, die mich ermutigt haben meine Promotion abzuschließen, als ich mit dem Gedanken spielte die Arbeit abzubrechen. Hier gilt mein besonderer Dank meinem Zweitgutachter Jörn Müller-Quade.

Nicht zuletzt bin ich meiner Familie, meinen Freunden, und insbesondere meiner Frau Lena unfassbar dankbar für die Unterstützung und den Rückhalt, den ich in dieser Zeit erfahren durfte. Ohne euch wäre diese Arbeit nicht möglich gewesen. Danke!

Contents

Acronyms

2DA Two Display Authentication

AAL Authenticator Assurance Level

AKE authenticated key exchange

ASLR address space layout randomization

ATM automated teller machine

AV antivirus

CAP chip authentication program

CDCVM Consumer Device Cardholder Verification Method

CTAP Client to Authenticator Protocol

DEP Data Execution Prevention

EDR endpoint detection and response

EMT electronic money transfer

EMV Europay, Mastercard, and Visa

FIDO Fast IDentity Online

FIDO2 FIDO2

FIDO2D FIDO2 With Two Displays

HSM hardware security module

MPC multi-party computation

mPIN mobile personal identification number

mTIN mobile transaction identification number

NFC near-field communication

OTP one-time password

PCS Post-Compromise Security

PFS perfect forward secrecy

PIN Personal Identification Number

POS point of sale

PPT probabilistic polynomial time

PSD2 revised Payment Services Directive

RBA risk-based authentication

SAGMA Secure Aggregation Grouped by Multiple Attributes

SE Secure Element

SSE searchable symmetric encryption

Tamarin Tamarin Prover

TAN transaction authentication number

TCG Trusted Computing Group

TEE Trusted Execution Environment

TLS Transport Layer Security

TPM Trusted Platform Module

TUI Trusted User Interface

txAuthSimple Simple Transaction Authorization Extension

U2F Universal Second Factor

UC Universal Composability

USB Universal Serial Bus

W3C World Wide Web Consortium

WebAuthn Web Authentication

Chapter 1

Introduction

The proliferation of the Internet provided unimaginable opportunities for society, people, and companies alike. Shopping online, connecting with people in a distant country or paying for goods on the other side of the globe has never been easier before. The interconnectedness of modern life also brings new challenges. Many IT systems are now accessible via the Internet and offer a huge potential for abuse.

Cryptography provides mechanisms to protect systems and data from unauthorized access. This includes encryption schemes to protect the confidentiality of data, digital signatures for authentication and many more. In a cryptographic protocol, multiple cryptographic primitives are combined to accomplish a specific goal. In the early days of cryptography, the design of cryptographic primitives and protocols was considered an art [111]. It was merely based on the experience of cryptographers as there was no established definition of what it means for a protocol to be secure. The field of cryptography evolved, and modern cryptographic protocols are designed based on a precise security definition that states requirements for a protocol to be secure and a formal proof of security that verifies that the protocol fulfills the security definition. This approach revolutionized the design of cryptographic protocols.

Nonetheless, provably secure schemes can be susceptible to attack when deployed in the real world. A proof of security is always limited to the aspects captured by the security notion and formal model. If the security notion does not depict the real-world accurately, then a provably secure protocol might be vulnerable to attack in practice. For example, the 4-way handshake used in the wireless network protocol WPA2 has been found to be susceptible to a devastating attack even though it was proven secure [156]. This was possible because the attack did not violate the security notion. The key installation, the critical protocol step for the attack, was not part of the formal model used for the security proof. Apart from weaknesses in the protocol itself, implementation errors can cause serious vulnerabilities too.

Protocols are usually modeled on an abstract level omitting implementation details. Thus, bugs in the implementation of a protocol are usually not captured by a security proof. Approaches that verify the correctness of the implementation of a cryptographic protocol exist [18, 117, 17]. However, they assume the correctness of the underlying hardware and thus verification of the full hardware and software stack is not accomplished. Furthermore, these approaches are very costly and are therefore not widely used [96]. Thus, the correctness of an implementation of a cryptographic protocol is usually considered a separate challenge that is tackled using automated and manual code review. Despite these shortcomings, formally proving a protocol secure provides a huge advantage. To analyze the security of a protocol in practice, one can compare the security definition to the situation in the real world and identify deviations [111]. These deviations might constitute possible ways to attack the protocol. A particularly important piece of the security definition, that we focus on in this work, is the attack model.

An attack model describes which capabilities an adversary is thought to have. In the 1980s, the Dolev-Yao attack model [54] was introduced and gained lots of popularity thereafter. This attack model assumes that the attacker has complete control of the network between computer systems and as such can read and manipulate all exchanged messages. However, the attacker cannot break cryptographic mechanisms without knowing the key.

An important threat which is not considered by the Dolev-Yao attack model is malware. When a device is infected by malware, it can be controlled by the adversary remotely. An adversary can either program the desired behavior into the malware and have it execute without manual intervention or control the malware manually by sending predefined commands. In the following, we use the terms *compromising a device* and *infection by malware* synonymously. There are several possibilities how a device can be infected by malware, e.g., drive-by downloads, malicious email attachments, and vulnerabilities in public-facing systems - just to name a few. In this work, we are not concerned with how an adversary gains access to a system and installs malware. Instead, we assume that infections happen regularly which is supported by related work. According to the 2022 Data Breach Investigations Report, every third data breach involved the use of malware [157]. In a surge of online banking malware in 2012, the European Union Agency for Cybersecurity (ENISA) even recommended to assume every customer device to be infected [64]. In our own research, we analyzed the security of multiple embedded devices and found numerous vulnerabilities. For example, we reviewed ten Battery Energy Storage Systems (BESS) and identified multiple vulnerabilities that ultimately allowed us to compromise three of the systems remotely over the network [16].

Modern operating systems provide protection mechanisms that isolate applications running in different user contexts that have individual privileges assigned to them. For example, only applications with administrative privi-

leges are allowed to change certain system settings or access restricted files. Thus, depending on the context the malware is running in, these mechanisms may limit the access the malware has on the system. However, malware usually runs in the context of the currently logged in user which provides various possibilities for malicious behavior. For example, in the Windows operating system it is possible to inject code into other applications that run in the same context such as browsers or password managers to manipulate their behavior or dump sensitive information[1]. Furthermore, malware has been found in the wild that contained privilege escalation exploits [68]. By using these exploits, an attacker gains higher privileges on the system. In this work, we do not differentiate between different user contexts on a system and assume malware to run with the highest privileges available. The advantage of this approach is that security guarantees are not invalidated by privilege escalation attacks.

Many cryptographic protocols, such as those based on the Dolev-Yao attack model, do not expect the adversary to install malware. This is particularly problematic, as malware with high privileges grants the adversary access to stored cryptographic keys and the ability to manipulate other programs running on the system. Thus, by gaining access to a computer, many security mechanisms can be rendered ineffective because they were never designed to protect against a local attacker on the system. For example, encrypting data in transit prevents an attacker controlling the network from learning the underlying plaintext, however, an attacker with access to the local system can just read the data before it is encrypted in the first place.

At a first glance, mechanisms used in online banking and electronic payment such as smart cards and two-factor authentication should protect against compromised systems. However, on closer inspection, many deficits come to light. For example, compromising an ATM is enough to issue fraudulent transactions and break the security of EMV, the world-wide standard for electronic payment. Even though EMV uses a smart card and an additional Personal Identification Number (PIN) to implement two-factor authentication, the protocol is susceptible to malware. The goal of this thesis is to improve the status quo and provide guidance for designers to create cryptographic protocols with resilience to malware attacks.

1.1 Research Question and Methodology

The main research question of this thesis is as follows:

> *How can cryptographic protocols be designed for real-world systems to provide resilience to attacks with malware?*

[1] `https://attack.mitre.org/techniques/T1055/`

To answer our research question, we investigate three real-world use cases: electronic payment systems, web authentication, and data aggregation. We choose these use cases based on their high potential for damage and as such high risk of malware attacks. In the following, we describe our methodology to analyze the use cases. For each use case we take the following steps:

(i) Security analysis of existing protocols

(ii) Requirements for malware resistance

(iii) Cryptographic protocol design

(iv) Formalization

(v) Proof of security

First, we assess the susceptibility of existing real-world protocols to malware attacks. Consequently, we identify requirements for a secure protocol that remedies the weaknesses identified in existing protocols. In particular, this includes determining the targeted security goals and attack model which make up the security notion. We then design a new protocol for the use case by relying on cryptographic and non-cryptographic mechanisms to achieve resilience to malware. Finally, we formalize the protocol and our security notion for malware-resistance and formally prove our protocol secure.

1.2 Contribution of this Work

This thesis takes important steps towards the use of malware-resistant protocols in real-world systems. We provide cryptographic protocols with malware resistance for multiple use cases, as well as security notions and formal models to verify their security. The contribution of each individual use case is described below.

1.2.1 Electronic Payment

EMV is the world-wide standard for electronic payment, which is used for withdrawing money at ATMs, as well as paying at the POS. A number of attacks demonstrated that compromising an ATM or POS device is sufficient to break the security guarantees of EMV [20, 57].

We state general requirements for secure electronic payment. The main challenge is to establish authenticated communication between a human user and the bank with possibly infected devices. As a remedy, we propose *one-out-of-two security*. This security notion states that a protocol must stay secure if at least one of two devices is not compromised. For example, a user's smartphone can be used as an additional device in the payment process.

We analyze current payment protocols and observe that they do not satisfy our requirements for secure electronic payment. In particular, they do not satisfy one-out-of-two security. Inspired by this analysis, we propose L-Pay, a secure protocol for electronic payment inspired by online banking protocols such as the chip authentication program (CAP) protocol and photoTAN. In L-Pay we require verification of a transaction on a user's smartphone and thereby protect against infected ATM and POS devices. However, our protocol also protects against an infected smartphone, as long as the ATM is not compromised.

We formalize L-Pay and our security notion *one-out-of-two security* using Tamarin Prover (Tamarin) [125], a formal verification tool for cryptographic protocols. Finally, we prove that L-Pay fulfills one-out-of-two security.

1.2.2 Web Authentication

The World Wide Web offers a plethora of services used by billions of people all around the world. Because the Web is used for many sensitive tasks such as online banking, shopping, and system administration, it is a lucrative target for attacks. We analyze the security of web authentication schemes in the presence of malware and real-time phishing attacks. We find that even strong authentication schemes for online banking that rely on *transaction authentication* as required by the revised Payment Services Directive (PSD2) do not protect against these attacks.

As a remedy, we propose that web authentication schemes handling security-critical transactions should fulfill *one-out-of-two security*, a security notion that neither requires a primary device nor an additional device trusted. We show how this security notion that we introduced for electronic payment can be adapted to web authentication. Furthermore, we provide a blueprint to design web authentication schemes that fulfill this notion and thus protect security-critical transactions even if one device is fully compromised.

Based on our blueprint, we design and implement FIDO2D, a web authentication scheme based on the FIDO2 standard. We identify shortcomings of FIDO2 for implementing malware-resistant web authentication and show how they can be treated. We examine multiple use cases on the Web that benefit from FIDO2D. Finally, we provide a formal model for FIDO2D and prove that it fulfills *one-out-of-two security* using Tamarin.

1.2.3 Data Aggregation

Data breaches have illustrated the danger of storing huge amounts of sensitive information in a central database. Cryptographic encryption schemes such as AES can be used to protect the confidentiality of stored data. However, this prevents query processing which is essential for several use cases such as data analytics.

We introduce SAGMA, a cryptographic protocol for secure aggregation that protects against compromised database servers. SAGMA fixes shortcomings of earlier approaches that rely on deterministic encryption to group data and are thus susceptible to leakage-abuse attacks. By splitting attribute values into buckets and hiding the frequencies of individual values in a bucket, SAGMA offers a high level of security. SAGMA supports aggregation queries with multiple grouping attributes. Furthermore, as aggregation is executed row-wise, it can be combined with additional filtering clauses as supported by searchable symmetric encryption (SSE). Hence, SAGMA can be integrated into encrypted database systems to support a wide variety of queries.

SAGMA provides semantic security and query execution only leaks the bucket membership of processed rows. We give guidelines on how the leakage can be further reduced using preprocessing of plaintext data. For performance reasons, the maximum number of grouping attributes in a single aggregation query has to be limited. We analyze aggregation queries used in real-world database applications such as Nextcloud, WordPress, and Piwik. Our analysis shows that this limitation does not restrict the applicability of our scheme to practical systems.

1.2.4 Overall Contribution

In a nutshell, this work paves the way for the adoption of malware-resistant protocols in real-world systems. We show that designing secure protocols that protect against malware is feasible. Even though there is no one-size-fits-all approach to secure protocol design, we provide guidelines that can be reused when designing protocols for other use cases. Our security notions show how malware-resistance can be formally defined. Together with our formal models we lay the foundations for future protocol design with malware-resistance.

1.3 Structure of this Work

The remainder of this thesis is structured as follows. In Chapter 2 we introduce preliminaries that serve as a foundation for the rest of the work. Chapter 3 covers our first use case based on electronic payment. In Chapter 4 we analyze our second use case that addresses web authentication. Our third use case on data aggregation is examined in Chapter 5. We summarize related work regarding malware-resistant protocols in Chapter 6. Finally, we conclude our work and provide perspectives for future work in Chapter 7.

Chapter 2

Preliminaries

In this chapter we introduce preliminary information that the following chapters are built upon. The informed reader may thus skip this chapter. We introduce basic protection goals, malware, cryptographic primitives and protocols, methods to formally prove the security of a cryptographic protocol, as well as secure data aggregation.

2.1 Protection Goals

In information security, protection goals describe security requirements of a system to limit access to data that is worth protecting [59]. The most common protection goals are *confidentiality*, *integrity*, *availability*, and *authenticity*. These protection goals are relevant for almost every system. We describe these main protection goals briefly below. The *confidentiality* of data is ensured if data is only accessible to authorized parties. To determine if a party is authorized to access data, one has to verify the identity of the party in the first place. This is covered by the protection goal *authenticity*. *Authenticity* is satisfied if it can be verified that the claimed identity of a party matches its real identity. The process of verifying the identity of a party is known as authentication. The *integrity* of data is ensured if data cannot be modified by unauthorized parties without being detected. Finally, the *availability* of data requires that an authorized user cannot be hindered to access data by an unauthorized party. For example, in a denial of service (DoS) attack, an attacker floods a system with a huge amount of traffic to affect the *availability* of the system.

Protection goals can be fulfilled by different means such as cryptographic primitives, access control mechanisms of operating systems, network firewalls and more. For example, the *authenticity* of data can be protected by using encryption. We introduce cryptographic primitives, such as encryption schemes, in Section 2.3. Our presented list of protection goals is far from complete. Various other protection goals have been proposed such as *non-*

repudiation and *privacy*. *Non-repudiation* means that a party cannot dispute that it has taken an action. This is for example important in digital contracts. *Privacy* ensures that an unauthorized party is not able to gather behavioral profiles of a system's user.

Protection goals are a fundamental tool for defining the security requirements of a system. In Section 2.5 we show how formal methods can be used to verify that a system fulfills certain protection goals.

2.2 Malware

Malware is an abbreviation for *malicious software*. It is a catchall term for programs that intentionally affect the confidentiality, integrity, or availability of a system [108]. Various types of malware exist, e.g., viruses, worms, trojan horses, ransomware, spyware, and wipers. These types define certain characteristics of the malware such as the propagation method or the impact. For example, worms are defined by its ability to spread to other devices in the network without assistance of the user. On the other hand, ransomware is defined by its impact on a system, namely encrypting data to extort money from the user. Not all types of malware are capable of spreading autonomously. Most of them require a mechanism to infect a system. The following infection vectors are frequently used to spread malware, e.g., malicious email attachments, drive-by downloads, social engineering attacks that trick the user into installing malware, infected removable media, and vulnerabilities in public-facing systems such as remote administration interfaces [153, 157].

Malware is a fundamental item in an attacker's toolbox. Once malware is running on a system, an attacker can achieve various goals. Malware can access sensitive data, log keystrokes to steal passwords, dump credentials stored on disk or in running processes, manipulate running processes or encrypt data for extortion [95, 24, 155, 157].

Modern operating systems isolate applications running with low privileges from applications running with higher privileges. For example, in the Windows operating system, each process has an associated integrity level [127]. Processes with low or medium integrity level are isolated from high-integrity processes and are not allowed to perform administrative tasks such as installing software on the system. Even for an administrative user, applications run with low or medium integrity by default. However, an administrative user can request to start an application with high integrity level by approving a User Account Control (UAC) prompt. Other operating systems provide similar mechanisms.

Commonly, malware runs in the context of the currently logged on user and as such does not have administrative privileges. However, this restricts the possibilities of an attacker running malware only slightly. First, there are

many opportunities for malware that do not require administrative privileges. For example, many sensitive applications such as a password managers or browsers run with the same privileges and can thus be accessed by malware. Banking malware has made use of this extensively to steal money from victims' bank accounts [24]. And second, there are many possibilities to escalate privileges and gain administrative privileges irregularly. As an example, vulnerabilities in the operating system can be exploited to gain higher privileges. Malware samples have been found in the wild that included privilege escalation exploits [68]. As such malware is a very potent threat to bypass security mechanisms and violate protection goals of a system.

2.3 Cryptographic Primitives

Cryptographic primitives are algorithms that serve as building blocks for more complex cryptographic protocols [9]. These primitives can be used to achieve certain protection goals such as authenticity, confidentiality, and integrity.

Encryption is probably the most widely known cryptographic primitive that is used to protect the confidentiality of data. In cryptography, a distinction is made between *private-key* (symmetric) and *public-key* (asymmetric) encryption. In private-key encryption, the same key is used for encryption and decryption. In contrast, in public-key encryption a key pair is generated that consists of separate public and private keys. The public key is used to encrypt data, while the private key can be used to decrypt a ciphertext. To give an example, RSA [145] is a famous public-key encryption scheme and AES [152] is a private-key encryption scheme that is widely used.

Public-key encryption can be applied in scenarios where the encrypting party should not have access to the private key. This is not possible with private-key encryption. However, private-key encryption usually exhibits superior performance and thus private-key and public-key encryption are often combined in a *hybrid* encryption scheme. Besides encryption many other cryptographic primitives exist. In the following, we introduce advanced cryptographic primitives that we use as part of our protocols in the following chapters.

2.3.1 Digital Signatures

Digital signature schemes are a cryptographic primitive that can be used to achieve authenticity. This primitive allows to *sign* a message with a private key that results in a *signature*. The message and the signature can be sent to another party that can verify that the message was really signed by the initiator. For this, the public key of the initiator has to be exchanged in advance.

A digital signature scheme consists of three algorithms as shown in Definition 1 [111]. Gen() is used to generate a key pair. Using Sign() one can generate a digital signature. Finally, Verify() allows to verify the validity of a signature. The following equation shows how these algorithms are intertwined. The signature of a message m created with the private key sk of a key pair should be successfully verified using the public key pk of the same key pair.

$$\text{Verify}(pk, \text{m}, \text{Sign}(sk, \text{m})) = 1$$

Definition 1. *A digital signature scheme consists of the following three algorithms:*

$(pk, sk) \leftarrow \text{Gen}(1^\lambda)$**:** *Generates a public/private key pair.*

$\sigma \leftarrow \text{Sign}(sk, m)$**:** *Creates a digital signature for message m using the private key sk.*

$b \leftarrow \text{Verify}(pk, \text{m}, \sigma)$**:** *Verifies a signature σ for a message m using the public key pk. Outputs $b = 1$ if the signature is valid and $b = 0$ otherwise.*

2.3.2 Homomorphic Encryption

Homomorphic encryption is a special type of encryption that enables a third party to perform calculations on encrypted data without having access to the private key. A famous example of homomorphic encryption is the textbook RSA scheme. Multiplying two ciphertexts of this scheme results in a ciphertext that contains the product of the plaintext values. This is illustrated by the following formula where $m^e \bmod N$ describes the regular RSA encryption of a message m.

$$(m_1)^e \bmod N \cdot (m_2)^e \bmod N \equiv (m_1 m_2)^e \bmod N$$

Similarly, the Paillier encryption scheme allows one to combine two ciphertexts such that the resulting ciphertext contains the sum of the plaintext values [134]. More formally, given two ciphertexts $[\![x]\!]_\oplus$ and $[\![y]\!]_\oplus$ one can compute the encrypted sum of the underlying plaintexts using the operation $\oplus$ on ciphertexts: $[\![x]\!]_\oplus \oplus [\![y]\!]_\oplus = [\![x + y]\!]_\oplus$. Such a scheme is said to be additively homomorphic. In homomorphic encryption, the encryption and decryption routines constitute homomorphisms between the plaintext and the ciphertext space, hence the name.

While many homomorphic encryption schemes only support one type of arithmetic operation, fully homomorphic encryption can be used to execute arbitrary computations on encrypted data. In his seminal work, Gentry introduced the first fully homomorphic encryption scheme [78]. However,

performance issues limit the applicability of fully homomorphic encryption in real-world systems to date.

Somewhat homomorphic encryption schemes bridge this gap by supporting more than one type of arithmetic operation on ciphertexts with some limitations but improved performance. For example, the BGN scheme supports addition and a single multiplication of ciphertexts [21].

2.3.3 Searchable Symmetric Encryption

With searchable symmetric encryption (SSE) it is possible to search for keywords in encrypted data [47]. For that purpose, an encrypted index is created. Without the used private key, no information can be extracted from the encrypted index (except for its size). The underlying data is structured as a document collection. A document collection is a collection of documents identified by unique document identifiers. Each document can consist of multiple keywords. To execute a search using the index, a search token is required. A search token for a specific keyword can be created using the private key. With the encrypted index and a search token, all documents can be searched for the keyword. However, a search operation only returns identifiers of matching documents and not any contents. The documents themselves can be encrypted with a regular encryption scheme and decrypted when necessary. We provide a formal description of an SSE scheme in Definition 2 (see [90]).

Definition 2. *A searchable symmetric encryption scheme consists of the following four algorithms:*

$K_{SSE} \leftarrow \mathrm{Gen}_{SSE}(1^{\lambda})$**:** *Generates a private key.*

$I \leftarrow \mathrm{Enc}_{SSE}(K_{SSE}, D)$**:** *Creates an encrypted index from a document collection D using the private key.*

$t_w \leftarrow \mathrm{Token}_{SSE}(K_{SSE}, w)$**:** *Creates a token t_w for a keyword w using the private key.*

$D(w) \leftarrow \mathrm{Search}_{SSE}(I, t_w)$**:** *Uses token t_w and an encrypted index I to search for documents containing the keyword w. Returns the document identifiers of all matching documents.*

Even though the search index is encrypted, searching in documents with search tokens leaks some information. In particular, the identifiers of documents that are returned from a search, also called *access pattern*, are accessible to an adversary. Furthermore, the *search pattern* reveals if two tokens search for the same keyword or not.

2.4 Cryptographic Protocols

A cryptographic protocol defines a series of steps and messages that are exchanged between participating parties to accomplish a specific functionality. In addition, cryptographic protocols aim to achieve certain protection goals. To fulfill these protection goals, cryptographic primitives are applied [19].

As an example, Transport Layer Security (TLS) [143] is a very common cryptographic protocol that establishes a secure channel between two parties and relies on various cryptographic primitives such as private-key encryption, public-key encryption, digital signatures and others. Below, we introduce the FIDO2 authentication protocol that we use as a foundation for our own protocol FIDO2D in Chapter 4.

FIDO2 FIDO2 is a set of standards published by the Fast IDentity Online (FIDO) Alliance and the World Wide Web Consortium (W3C) [4]. It defines a challenge-response authentication protocol using public-key cryptography. As such FIDO2 can be used to establish the authenticity of a user. FIDO2 is mainly used on the Web via the HTTP protocol, however, some other protocols also support FIDO2 authentication, for example SSH. FIDO2 can either be used as a second factor for password-based authentication, or without a password in passwordless mode.

FIDO2 defines two ceremonies, one for registration and one for authentication. A *client* initiates a registration or authentication ceremony with a service (also called *relying party*). Cryptographic operations are executed by an *authenticator* that can either be an external device or integrated into the platform that runs the FIDO client. FIDO2 uses digital signatures and a challenge-response technique for authentication. During registration the client transmits the public key of a fresh key pair to the relying party. The key pair is generated by the authenticator and bound to the specific relying party. Thereby, phishing attacks are prevented as a cryptographic key can only be used for a specific relying party identified by its domain. For authentication, the authenticator has to sign a challenge supplied by the relying party and forwarded by the client. The generated signature is forwarded by the client to the relying party. Authentication succeeds if the relying party verifies the digital signature successfully.

Figure 2.1 shows a simplified flow of the FIDO2 registration and authentication ceremonies. During registration, the client forwards a challenge provided by the relying party, coupled with the identifier of the relying party (*rpId*). The authenticator generates a key pair and responds with the public portion of the key pair, which is again forwarded by the client to the relying party. In consequence, the relying party stores the public key and associates it with the user's account. The authentication ceremony works very similar. Again, the relying party sends a challenge to the client, which is forwarded to the authenticator. During authentication, the authenticator responds with

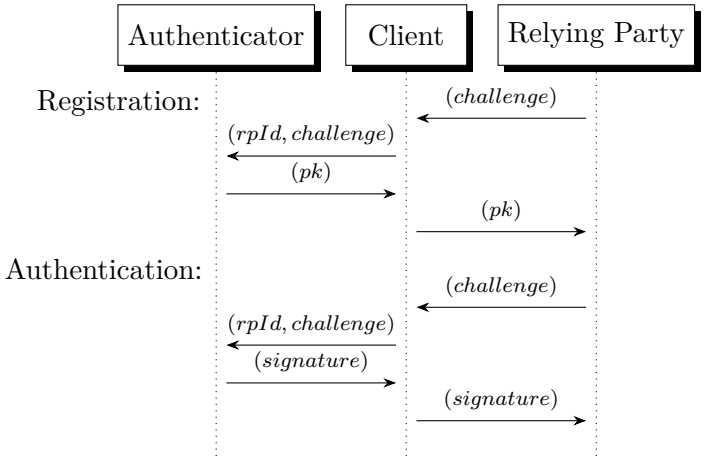


Figure 2.1: Simplified protocol flow of FIDO2 registration and authentication.

a signature instead of a public key. The signed data includes the *rpId* and the challenge provided by the relying party. The relying party then retrieves the stored public key of the account and verifies the digital signature. If the signature is valid, the authentication ceremony succeeds.

FIDO2 consists of two standards; WebAuthn is a JavaScript API supported by current browsers and Client to Authenticator Protocol (CTAP) defines the communication between the FIDO client and an external authenticator [164, 69]. The standards do not require a specific server-side API and thus exact details of the communication between the client and the relying party are up to the implementer. However, the communication between the client and the relying party must be protected using TLS.

FIDO ceremonies require a test of *user presence*, i.e., the authenticator must verify that the user is present before fulfilling a request. For this, authenticators usually provide a hardware button that has to be touched by the user. Furthermore, some authenticators support to authenticate a user based on a PIN or biometric features such as a fingerprint scan. This feature is called *user verification* and has to be requested by the relying party for a ceremony explicitly.

2.5 Provable Security

Using formal methods to prove the security of cryptographic primitives and protocols is the central concept of modern cryptography [111]. In contrast to former practices, the designer defines when a scheme is considered secure using a security definition. This includes the assumed threat model, i.e., the capabilities of the adversary, as well as the targeted security guarantees. Then, a rigorous proof of security is conducted to verify that a cryptographic

primitive or protocol fulfills the definition and can thus be considered secure. A major advantage of this approach is that the targeted security guarantees are precisely defined, and additional assumptions are explicitly documented.

Various techniques for modeling and verifying the security of cryptographic protocols exist. They all have their own strength and weaknesses. Originally, the security of cryptographic schemes has been verified using pen and paper. For example, game-based proofs have been used extensively to verify the security of cryptographic primitives. For complex cryptographic protocols, simulation-based proofs are more convenient. Nowadays, the use of computer-assisted proof systems such as Tamarin is on the rise. These proof systems support the user by automating the proof generation. Thus, the user only has to define the security definition and formalize the protocol.

In the following, we introduce three relevant techniques for the formal verification of cryptographic protocols: simulation-based security, Universal Composability (UC), and Tamarin a tool for the computer-assisted verification.

2.5.1 Simulation-based Security

Determining an appropriate security definition for a cryptographic protocol is a difficult task. This is for example highlighted by the sheer amount of security definitions that exist for secure encryption such as *semantic security*, *indistinguishability under chosen plaintext attack*, and *indistinguishability under chosen ciphertext attack*. One solution to this problem is simulation-based security, also known as the *real-ideal paradigm*. In this approach, security is defined using a comparison between a real-world and an ideal-world experiment [119]. In the real experiment, the protocol is executed in the presence of an adversary $\mathcal{A}$. In contrast, in the ideal experiment, the protocol execution is simulated by a simulator $\mathcal{S}$ and is secure by definition. For example, in an ideal world for secure encryption, the simulator is not given a ciphertext, but only the length of the plaintext data. However, if the simulator is able to simulate the output of the adversary in the real experiment, then the adversary was not able to learn anything beyond the length of the plaintext.

More formally, the security of a protocol can be defined as shown in Definition 3 where PPT stands for probabilistic polynomial time (see [47] for more details). This security definition states that a protocol is considered secure if the outputs of the real and the ideal experiment are indistinguishable. This means that for all attacks that affect the real experiment, there must exist a simulator $\mathcal{S}$ that has the same effect in the ideal world, which is secure by definition.

Definition 3 (Simulation-based security)**.** *A cryptographic scheme is secure, if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for all PPT algorithms $\mathcal{D}$ it holds that*

$$\begin{aligned}&\big|\Pr\left[\mathcal{D}(v, st_{\mathcal{A}}) = 1 : (v, st_{\mathcal{A}}) \leftarrow \mathbf{Real}_{\mathcal{A}}(\lambda)\right] \\ &\quad - \Pr\left[\mathcal{D}(v, st_{\mathcal{A}}) = 1 : (v, st_{\mathcal{A}}) \leftarrow \mathbf{Sim}_{\mathcal{A},\mathcal{S}}(\lambda)\right]\big| \leq \mathrm{negl}(\lambda)\end{aligned}$$

where v defines the view of the adversary, $st_{\mathcal{A}}$ is the state of the adversary, and $\mathrm{negl}(\lambda)$ *is negligible in λ.*

2.5.2 Universal Composability

Simulation-based security has been further extended in the UC framework [27]. Cryptographic primitives that are proven secure using a security definition in the UC framework provide an additional guarantee, i.e., they are *universally composable*. That means, a primitive can be used in a larger system with an arbitrary number of other primitives while preserving the guarantees of the security notion. Other security notions, e.g., game-based ones do not have this property. Some primitives have been shown to be vulnerable when they are used in a larger system. For example, encryption schemes that achieve semantic security, a classic security definition for encryption, can be broken when they are used in a scenario where the attacker is able to send ciphertexts to a system and get feedback on whether decryption was successful. This even led to many real-world vulnerabilities [126].

The composition property of the UC framework allows to break down security proofs for larger systems into smaller components. Cryptographic primitives can be proven secure individually and the composition property ensures that they can be integrated securely. However, this strong guarantee comes at a price. Even basic functionalities cannot be proven secure in the UC model without additional assumptions [28]. Furthermore, creating a security proof is a manual and error-prone task.

2.5.3 Tamarin

Tamarin is a tool for the computer-assisted verification of cryptographic protocols [149, 154]. It has been used to verify security properties of various real-world protocols. For example, Tamarin has been used to prove security properties of TLS 1.3 [45]. Protocols are modeled in Tamarin using *multiset rewriting rules*. These rules describe a protocol's execution flow and define a labeled transition system that modifies a global state consisting of a multiset of *facts*. Security properties can be expressed as first-order formulas that are verified against the traces of the transition system. Messages can be exchanged between parties of the protocol using a shared network. By default, Tamarin assumes a Dolev-Yao [54] attacker that has full control of

the network. Tamarin explicitly models the knowledge of the attacker, which is automatically extended by messages sent over the network.

Once the protocol and the security properties are modeled, Tamarin can construct proofs automatically. To do this, Tamarin uses deduction and equational reasoning. Proofs consider an unbounded number of protocol sessions. Furthermore, Tamarin can provide a counterexample that represents an attack on the protocol if the protocol cannot be proven secure. However, it is not guaranteed that the prover terminates, since the problem of verifying cryptographic protocols is undecidable [149]. In this case, it is possible to use the interactive mode to manually guide the proof construction process instead of relying on the heuristics that are used in the automatic mode.

In the following, we give a brief introduction to the Tamarin modeling language. For more details, we refer to the Tamarin manual [154]. Multiset rewriting rules consist of three parts: a left-hand side of input facts that are consumed by the rule, a right-hand side of output facts that are added to the global state once the rule has been applied and a set of action facts that are written to the trace. An exemplary rule is shown below:

```
rule example:
    [ INPUT_FACTS ]
    --[ ACTION_FACTS ]->
    [  OUTPUT_FACTS ]
```

Facts consist of a name and a fixed number of terms. For example, the fact `!Pk($I, pk)` uses two terms to map an identifier to a public key. There are two types of facts: linear facts that are removed from the global state once they are consumed by a rule and persistent facts prefixed with an exclamation mark that can be consumed by rules arbitrarily often.

Multiset rewriting rules can be used to model state transitions of the cryptographic protocol. Rules can be chained by outputting a fact from one rule that is consumed by another rule. Initially, the global state is empty. Thus, a model usually contains at least one rule with an empty set of input facts that can be applied initially. Action facts are not added or removed from the global state but written to the execution trace. They are relevant for defining security properties that can only refer to action facts.

In Tamarin's modeling language, the type of a variable is determined by its prefix. Once a variable has been defined, the prefix may be omitted in subsequent use. We describe the main types briefly below.

- `~x` denotes a *fresh* variable which is guaranteed to have a unique value.
- `$x` denotes a *public* variable. Public variables can share the same value.
- `#i` denotes a *temporal* variable. Temporal variables are mainly used in security definitions.
- `m` denotes a *message* variable. A message is either public or fresh.

Tamarin offers special facts that are generated by built-in rules and provide functionalities such as drawing random numbers and accessing the network. We describe these special facts briefly below.

- `Fr(~r)` Introduce a fresh variable `r`. Two fresh variables are guaranteed to have different values. This can be used to model a nonce or a cryptographic key. Fresh variables can only be introduced using the `Fr()` fact.
- `In(m)` Read a message m from the network. Messages sent to the network can be read, manipulated, and replayed by the adversary arbitrarily.
- `Out(m)` Write a message m to the network. This can be used to leak information to the attacker deliberately.
- `K(m)` Models that the adversary has knowledge of m.

Security properties are defined as *lemmas* in Tamarin. They are expressed as first-order formulas that are verified against the traces of a model. We sketch the syntax for lemmas in Table 2.1, which is based on the Tamarin manual [154]. We show an exemplary lemma below, that states that every trace that contains the fact F() must also contain the fact G().

```
lemma example:
    "All m #i. F(m) @i ==> (Ex n #j. G(n) @j)"
```

Security properties usually begin with a universal quantifier to state that all traces must fulfill a certain property. However, lemmas cannot only be used to verify security properties but also to do sanity checks of a model. These lemmas usually begin with an existential quantifier. For example, they can be used to verify that a certain state in the modeled protocol can be reached. These lemmas are also called *sanity* lemmas.

Tamarin provides built-in message theories for various cryptographic primitives. For example, it provides mechanisms that model digital signatures, private-key encryption, hashing, and public-key encryption. Digital signatures can be used by including the *signing* package. It provides four functions $\text{sign}(m, sk)$, $\text{verify}(s, m, pk)$, $\text{pk}(sk)$, and true. The function pk() generates a public key from a secret key. The function sign() can be used to create a signature and verify() to verify a generated signature. Key generation can be modeled by using the built-in Fr() fact. These functions are associated by the following equation:

$$\text{verify}(\text{sign}(m, sk), m, \text{pk}(sk)) = \text{true}$$

Table 2.1: Syntax for defining lemmas in Tamarin as defined in [154].

`All`	universal quantification
`Ex`	existential quantification
`==>`	implication
`&`	conjunction
`|`	disjunction
`not`	negation
`f @ i`	action constraints
`i < j`	temporal ordering
`#i = #j`	equality of temporal variables
`x = y`	equality of message variables

Similarly, asymmetric encryption can be included using the *asymmetric-encryption* package. It provides three functions $\text{aenc}(m, pk)$, $\text{adec}(c, sk)$, and $\text{pk}(sk)$ for encryption, decryption and derivation of a public key from a private key. These functions are associated by the following equation:

$$\text{adec}(\text{aenc}(m, \text{pk}(sk)), sk) = m$$

In Tamarin, a *restriction* can be used to specify additional requirements that must be fulfilled by a trace to be considered valid. For example, when using digital signatures, a restriction is used to eliminate traces where the signature verification failed. This is implemented by defining a fact Eq that ensures that both its inputs are equal. This restriction is shown below:

```
restriction Equality:
    "All x y #i. Eq(x, y) @i ==> x = y"
```

2.6 Data Aggregation

In this section, we introduce preliminary information on secure data aggregation that we need as a basis for Chapter 5. The contents of this section are based on joint work with Florian Hahn and Florian Kerschbaum. Parts of this section previously appeared in the following publications.

- Timon Hackenjos. "Secure Aggregation and Grouping in Encrypted Databases". MA thesis. Karlsruhe Institute of Technology (KIT), 2017 [89]
- Timon Hackenjos, Florian Hahn, and Florian Kerschbaum. "SAGMA: Secure aggregation grouped by multiple attributes". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 587–601. DOI: 10.1145/3318464.3380569 [90]

ID	Salary	Gender	Name	Department
1	1000	male	Henry	Sales
2	5000	female	Jessica	Sales
3	1500	female	Alice	Finance
4	3000	male	Bob	Sales
5	2000	male	Paul	Facility

Table 2.2: Example table supported by our construction.

Aggregation is one of the most common types of queries in data analytics. The goal of secure data aggregation is to allow an untrusted database server to aggregate stored data while still preserving the confidentiality of the data. Conventional encryption schemes as introduced in Section 2.3 are not suitable for this scenario, as they prevent any type of processing on the server. Data aggregation is especially challenging since it is commonly combined with grouping functionality that requires to aggregate data in groups based on another attribute's value [90]. This problem has been approached in the past using property-preserving encryption [109]. However, it has been shown that property-preserving encryption is prone to inference attacks [130]. In the following, we describe data aggregation with grouping in more detail and present the solution from [90] that we extend in Chapter 5.

2.6.1 Problem Description

In the remainder of this section, we assume a relational database table with the following general layout (we use the terms column and attribute interchangeably):

1. We assume at least one column, called *value column*, to be aggregated (e.g., summed, counted, averaged),

2. further, we assume one or multiple attributes, called *group columns*, the GroupBy clause is executed on,

3. and finally, we assume zero or multiple auxiliary attributes, called *filtering columns*, additional filtering clauses are executed on.

One can define group and value columns as filtering columns as well. We give an example sketched in Table 2.2 to demonstrate this setting. Here, "Salary" is a value column, "Gender" and "Department" are group columns and "Name" and "Department" are filtering columns. A possible query formulated in SQL is given in Listing 2.1 yielding the corresponding result depicted in Table 2.3.

We aim for an approach that supports these kinds of queries and protects against a persistent adversary compromising the database. For example,

```
SELECT SUM(Salary), Gender, Department
FROM Example
WHERE Department = ''Sales''
GROUP BY Gender, Department;
```

Listing 2.1: Example SQL query

SUM(Salary)	Gender	Department
5000	female	Sales
4000	male	Sales

Table 2.3: Result of executing SQL from Listing 2.1 on Table 2.2.

an adversary might infect a database server with malware to dump the database and monitor database operations. We revise previous approaches before we outline SAGMA and briefly highlight the differences. We refer to Section 5.2.3 for a thorough comparison.

One approach for secure aggregation in previous work reveals the access pattern. For example, CryptDB [138] supports data aggregation with arbitrary GroupBy attributes and combinations thereof on encrypted data. The evaluation of the GroupBy clauses is based on deterministic encryption of values in group columns and additively homomorphic encryption of values in value columns. Deterministic encryption $\text{Enc}_{det}(\cdot)$ preserves equality: given two plaintexts a and b with $a = b$ it holds that $\text{Enc}_{det}(a) = \text{Enc}_{det}(b)$. We denote the encryption of a plaintext value x using an additively homomorphic encryption scheme by $[\![x]\!]_{\oplus}$. Given two ciphertexts $[\![x]\!]_{\oplus}$ and $[\![y]\!]_{\oplus}$ one can compute the encrypted sum of the underlying plaintexts using the operation $\oplus$ on ciphertexts: $[\![x]\!]_{\oplus} \oplus [\![y]\!]_{\oplus} = [\![x + y]\!]_{\oplus}$. In combination, the DBMS can perform GroupBy operations on encrypted data, e.g., grouping by $\text{Enc}_{det}(a)$ and subsequently perform aggregation operations for each such group using the operation $\oplus$. While additively homomorphic encryption offers semantic security, the security of deterministic encryption is questionable. Deterministic encryption leaks joint group membership for all rows enabling an attacker to reconstruct a histogram of all group values. In many cases, this histogram enables simple, yet powerful attacks as shown by Naveed et al. [130].

An alternative approach with the goal to address this security issue for aggregation is to compute (and encrypt) an index over the group columns as proposed by Papadimitriou et al. with Seabed [135]. Seabed also uses a combination of deterministic and additively homomorphic encryption. However, by splitting group attributes into multiple columns and introducing dummy elements they flatten the leaked plaintext frequencies thwarting frequency analysis. Compared to CryptDB, this approach provides better security for the grouping values; however, it restricts the choice of attribute

combinations in one GroupBy statement. Further, statements containing additional filtering clauses in combination with data aggregation requires computation effort by the client linear in the filtering result size.

A naïve strategy is to pre-compute the aggregation results for combinations of group column values initially during encryption time. However, there are an exponential number of such combinations of group column values requiring excessive storage space. Considering additional filtering statements, the number of potential results to be pre-computed becomes even more impractical.

Our scheme SAGMA does not unveil the access pattern for individual grouping values. At the same time, SAGMA enables queries with GroupBy clauses over multiple group columns and arbitrary number of value attributes together with support for additional filtering statements. In detail, SAGMA employs row-wise encryption such that it can be easily combined with searchable encryption schemes, e.g., supporting filtering for specific keywords [47], ranges or substrings [67, 92] and is even compatible with complete systems [109]. Hence, it is possible to process queries by first executing the filtering operation and then using SAGMA on the result set for secure aggregation. We emphasize that this filtering is not feasible for secure aggregation schemes based on pre-built indexes or pre-computed results.

2.6.2 Constructions

In this section, we develop the ideas for our construction of SAGMA step-by-step before we give a formal description of these ideas in Section 2.6.3. Our constructions use ciphertext packing initially proposed by Ge et al. [77] for performance improvements of additively homomorphic encryption. We divide the plaintext into several blocks enabling encryption of multiple values in one ciphertext. One can determine these blocks either during the initial encryption step or during the actual data aggregation as elaborated in the following subsections. Applying homomorphic addition allows us to add values componentwise.

Initial Static Shifting

We re-use the idea of ciphertext packing, however, instead of increased performance we aim for increased security. Basically, each ciphertext consists of multiple blocks and during aggregation each block contains the current subtotal for one group attribute value. More specifically, we interpret the plaintext space $\mathcal{M}$ of an additively homomorphic encryption scheme as multiple separate *value blocks* with value domain D_V. For example, using secure parameters the plaintext space $\mathcal{M}$ of the additively homomorphic encryption scheme published by Pailler [134] has a size of 2048 bits, and a value domain D_V has a size of 32 bits corresponding to the common integer

0	1000	$\Rightarrow$	$[\![1000]\!]_{\oplus}$
5000	0	$\Rightarrow$	$[\![5000 \cdot 2^{32}]\!]_{\oplus}$
1500	0	$\Rightarrow$	$[\![1500 \cdot 2^{32}]\!]_{\oplus}$
0	3000	$\Rightarrow$	$[\![3000]\!]_{\oplus}$
0	2000	$\Rightarrow$	$[\![2000]\!]_{\oplus}$

Figure 2.2: Example encoding for tuples consisting of value and group attributes as given in Table 2.2.

size. We encode the value v and a group attribute g (of small sized group attribute domain D) in a transformed value v' to be encrypted afterwards: The group attribute value is encoded by the index of the block containing value v. All remaining blocks are set to zero. Given the group attribute domain, e.g., $D = \{\text{male}, \text{female}\}$ as sketched in Table 2.2 and value domain $D_V = \{0, \ldots, 2^{32} - 1\}$ we can encode the tuples $(1000, \text{male})$, $(5000, \text{female})$, $(1500, \text{female})$, $(3000, \text{male})$, $(2000, \text{male})$ as given in the following Figure 2.2, where each block has bitlength 32.

From a mathematical point of view, we use a *mapping function* $f : D \rightarrow \{0, \ldots, |D|\}$ mapping group attributes to positive integers and use f to determine the *blockwise left shift* s encoding the group membership of value $v \in D_V$ into one transformed value v' by multiplication

$$v' = v \cdot s(g) = v \cdot |D_V|^{f(g)}.$$

This transformed value v' is then encrypted using an additively homomorphic encryption scheme resulting in ciphertext $[\![v']\!]_{\oplus}$. The sum over all data in combination with GroupBy statement is then transformed to a general aggregation over encrypted data computed by additive homomorphic encryption. The client can decrypt the result and extract the individual group totals by extracting the corresponding block. This approach increases security for individual group values as it hides their access pattern.

While this scheme inherits the security of the additively homomorphic encryption scheme, the group attribute domain size is restricted: the number of distinct group attribute values must be smaller than the maximum number of value blocks fitting in the plaintext domain, i.e., $\left\lceil \frac{|\mathcal{M}|}{|D_V|} \right\rceil \geq |D|$.

This constraint can be addressed by concatenating multiple ciphertexts and executing homomorphic addition componentwise. That is, given an additively homomorphic encryption scheme with plaintext size $|\mathcal{M}|$ divided in b blocks (i.e. $\left\lceil \frac{|\mathcal{M}|}{|D_V|} \right\rceil = b$), hence supporting group attribute domains up to size b, we can extend the size by factor n by concatenating n plaintext messages each encrypted separately, i.e., $m' = m_1, m_2, \ldots, m_n \in \mathcal{M}^n$. As one major drawback, however, this construction results in additional storage overhead where most parts of m' contain zeros but still provides semantic security.

Statically Shifted Bucketization

To reduce the storage overhead of our initial construction, we divide the complete group attribute domain D in separate buckets each with bucket size B. Hence, each transformed value only consists of B instead of $|D|$ blocks and thus requires less storage space. Aggregation is performed in each bucket separately but discloses the bucket membership for each row. Still, values of the same bucket are indistinguishable for an adversary.

To map the group attribute value to one of the B blocks, we use a simple modulo operation. The value v is shifted to the appropriate block by multiplying it by the shift $s(g) = |D_V|^{f(g) \bmod B}$. Here, the mapping function f can be seeded with an additional secret key, preventing (and hiding) coherent group values to be mapped to the same bucket.

Note that the bucket membership of rows itself can be protected, e.g., using searchable symmetric encryption with support for Boolean search queries such as published by Cash et al. [29]. At least for databases where the aggregation is only computed over a subset of the complete outsourced data set determined by additional filtering attributes. This supplementary protection unveils the same bucket membership only for rows matching the additional filtering clause. The bucket size B is an additional parameter providing the possibility to trade security for required storage space for each encrypted value as well as computation time since the aggregation is executed componentwise. We give a more detailed security analysis on the bucket size B and bucketing strategies in Section 5.2.1.

Dynamically Shifted Bucketization

So far, we have only addressed secure aggregation protocols for databases containing a single value attribute to be aggregated, however, we strive to generalize our construction to provide functionality for multiple columns to be aggregated. For simplicity, we assume them to have the same value domain D_V. This construction can also be used for additional aggregation functionality, e.g., count queries can be supported by encrypting value attributes fixed to one. The construction described before can be extended canonically to support multiple value attributes by repeated application for each value column. However, the group membership is then encoded in each value separately, resulting in redundant information. In order to achieve better storage efficiency, we store the value attributes and the shift values $s(g_i)$ determined by the group value g_i separately and multiply them on the server when queried. Since these values are assumed to be sensitive, they have to be stored encrypted in a way still supporting multiplication (over ciphertexts). This can be realized using somewhat homomorphic encryption (SWHE) $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec}, \oplus, \otimes)$ supporting one single multiplication and being additively homomorphic even after multiplication [21].

E_Salary	E_Gender	...
$[\![1000]\!]^{\otimes}_{\oplus}$	$[\![1]\!]^{\otimes}_{\oplus}$	
$[\![5000]\!]^{\otimes}_{\oplus}$	$[\![2^{32}]\!]^{\otimes}_{\oplus}$	
$[\![1500]\!]^{\otimes}_{\oplus}$	$[\![2^{32}]\!]^{\otimes}_{\oplus}$	
$[\![3000]\!]^{\otimes}_{\oplus}$	$[\![1]\!]^{\otimes}_{\oplus}$	
$[\![2000]\!]^{\otimes}_{\oplus}$	$[\![1]\!]^{\otimes}_{\oplus}$	

Table 2.4: Encrypted table with dynamic shifts.

We use the notation $[\![x]\!]^{\otimes}_{\oplus}$ to describe the encryption of a plaintext x using algorithm Enc of a SWHE scheme. The operations $\oplus$ and $\otimes$ denote the additively homomorphic and multiplicatively homomorphic operation. We denote the multiplication of a ciphertext $[\![x]\!]^{\otimes}_{\oplus}$ by a plaintext y as $[\![x]\!]^{\otimes}_{\oplus} \otimes y = [\![x \cdot y]\!]^{\otimes}_{\oplus}$. This operation uses the additively homomorphic property of the encryption scheme only and should not be confused with the multiplicatively homomorphic operation that multiplies two ciphertexts. Given the previous example in Table 2.2, the transformed table using dynamically shifted bucketization is given in Table 2.4 with $s(\text{male}) = 1$ and $s(\text{female}) = 2^{32}$.

To generalize the construction for multiple attributes, we evaluate the shift $s(g_i)$ of a group attribute value g_i on the server. We describe the shift as polynomial and determine the coefficients a_i such that $P(x) = (|D_V|)^x = s(g_i)$ for $x \in \{0, \ldots, B-1\}$ given by $x = f(g_i) \bmod B$ with mapping function f. Recall, that ciphertexts of a somewhat homomorphic encryption scheme can be multiplied by a plaintext value. Thus, the server can evaluate polynomials given plaintext coefficients a_i of P and encrypted monomials $[\![1]\!]^{\otimes}_{\oplus}$, $[\![x]\!]^{\otimes}_{\oplus}$, $[\![x^2]\!]^{\otimes}_{\oplus}$, ..., $[\![x^{B-1}]\!]^{\otimes}_{\oplus}$, where the degree of this polynomial increases linearly with the bucket size B. We transfer the coefficients to the server during encryption of the database to reduce the network bandwidth of aggregation queries.

Multiple Grouping Attributes

Based on the previous ideas we describe our final SAGMA construction supporting GroupBy statements with multiple grouping attributes in the same query as stated in Section 2.6.3. Referring to the example from Table 2.2 this construction supports statements grouping by "Gender" and "Department" or arbitrary subsets thereof, e.g., solely GroupBy "Department".

Naïve scheme While inefficient, this can be implemented using the previous scheme with the power set of group attributes. Note, that it is necessary to use a combined bucket size of B^i for a subset of i attributes in order to avoid additional leakage. Assuming two attributes and the bucket size $B = 2$, we demonstrate a potential attack where an adversary can query a GroupBy operation for both attributes separately and their combination.

ID	Gender	Department	Gender, Department
1	Gen_1	Dept_1	GenDept_1
2	Gen_1	Dept_1	GenDept_2

Table 2.5: Possible bucket memberships for Table 2.2.

These queries leak the bucket membership for both attributes individually and the combination, as illustrated in Table 2.5. Based on the two individual attributes, the two rows are indistinguishable since they are part of the same buckets. However, the two rows do not contain the same values, thus they might be mapped to separate buckets of the combined attribute. Mapping them to separate buckets leaks the fact that these two rows do not contain the same values. Both buckets Gen_1 and Dept_1 contain two values, therefore there are four possible value combinations of the two buckets. To prevent this leakage, all value combinations of the two buckets are mapped to the same bucket of the combined attribute requiring a bucket size of $B = 4$. More generally, an attribute combination of i attributes requires a bucket size of B^i to achieve the same leakage as in the single attribute case.

Improved scheme We use the polynomial approach described in Section 2.6.2 and extend it to multivariate polynomials where one variable represents one group attribute. The shift for a combination of attributes $G_1, \ldots, G_l$ can be determined by the polynomial of l variables:

$$P(G_1, \ldots, G_l) = \sum_{i_1,\ldots,i_l} a_{i_1,\ldots,i_l} \cdot G_1^{i_1} \cdots G_l^{i_l}.$$

We improve the naïve scheme by reusing the monomials required for individual attribute grouping for the grouping of attribute combinations. More generally, to group a set of attributes all monomials of the subsets can be used, thus reducing the number of monomials required to be stored on the server. Due to the empty product, only $B - 1$ monomials are necessary for a single attribute using bucket size B.

Considering three group attributes G1, G2, and G3 we demonstrate the difference. The naïve scheme requires one monomial for each of the individual attributes G1, G2, and G3, three monomials for the attribute combinations of size two and seven monomials for the combination of all three attributes. The improved scheme also requires one monomial for each individual attribute; however, the attribute combinations of size two only require one additional monomial because the two monomials of the individual attributes can be reused.

The idea of monomial reuse is sketched in Figure 2.3; here the beginning of an arrow represents monomials required to support grouping by a specific attribute combination where monomials can be re-used for the attribute

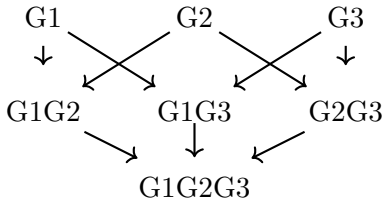


Figure 2.3: Monomial reuse that supports multiple attributes.

combination denoted at the arrowhead. This relation is transitive, i.e., the attribute combination of all three attributes can reuse the monomials of all its subsets, thus only requiring one additional monomial. In this case, we can reduce the required number of monomials from nineteen to seven.

Based on Table 2.2 we give an example that demonstrates grouping of multiple attributes. In this example, we assume bucket size $B = 2$ resulting in the following buckets: one bucket for the "Gender" attribute named Gen_1 containing {male, female} and two buckets for the "Department" attribute named Dept_1 containing {Sales, Finance} and Dept_2 containing {Facility}. Bucket membership for each row is indexed using searchable symmetric encryption as sketched [1] in Table 2.6.

Bucket	Rows
Gen_1	$1, 2, 3, 4, 5$
Dept_1	$1, 2, 3, 4$
Dept_2	5

Table 2.6: Bucket index supporting multiple attributes.

In our use case for SSE, the encrypted index is created for all bucket identifiers of the complete table consisting of multiple rows. Particularly, the document collection D then corresponds to the complete table and one document corresponds to one specific row, where each such "document" contains its specific bucket identifiers as searchable keywords. Using SSE, rows belonging to a specific bucket can only be determined using a token generated by the client.

Assume mapping functions $f_1(\text{male}) = 0$, $f_1(\text{female}) = 1$ and $f_2(\text{Sales}) = 0$, $f_2(\text{Finance}) = 1$ and $f_2(\text{Facility}) = 2$. Note that each value is reduced mod B before encryption. Further, in order to support grouping by both attributes Gender and Department, the client must generate and outsource an additional monomial, namely their product. This results in the encrypted and outsourced data sketched in Table 2.7.

[1] This is a sketch for demonstration purpose only and constructing efficient SSE indexes is its own line of research.

ID	E_Salary	E_Gender	E_Department	E_Gender · Dept
1	$[\![1000]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$
2	$[\![5000]\!]^{\otimes}_{\oplus}$	$[\![1]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$
3	$[\![1500]\!]^{\otimes}_{\oplus}$	$[\![1]\!]^{\otimes}_{\oplus}$	$[\![1]\!]^{\otimes}_{\oplus}$	$[\![1]\!]^{\otimes}_{\oplus}$
4	$[\![3000]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$
5	$[\![2000]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$	$[\![0]\!]^{\otimes}_{\oplus}$

Table 2.7: Encrypted table with multiple attributes.

We determine a multi-variate polynomial $P(G_1, G_2)$ that maps combinations of attribute values to the proper shift, using a system of linear equations. In the following, G_1 represents the attribute Gender and G_2 the attribute Department. However, the solution of the linear system can be reused for other attributes using the same bucket size.

$$\begin{matrix} 1 & G_1 & G_2 & G_1 \cdot G_2 \end{matrix}$$
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2^{32} \\ 2^{64} \\ 2^{96} \end{pmatrix}$$

One solution for the system is $a_0 = 1$, $a_1 = 2^{64} - 1$, $a_2 = 2^{32} - 1$, $a_3 = 2^{96} - 1 - (2^{64} - 1) - (2^{32} - 1)$. These coefficients are transferred to the server in plaintext during the encryption phase. Later, the client requests to execute the query from Listing 2.2 grouping by both attributes.

```
SELECT SUM(Salary), Gender, Department
FROM example
GROUP BY Gender, Department;
```

Listing 2.2: Query grouping by two attributes.

Therefore, the client determines SSE tokens for the buckets Gen_1, Dept_1 and Dept_2 to send them to the server in addition to the identifier of the attribute Salary to be aggregated and the identifiers of the attributes Gender and Department to be grouped. The server uses these tokens to determine the rows that belong to a specific bucket. Then, by calculating the intersection, the server determines the rows that belong to a specific bucket combination. Alternatively, an SSE scheme that supports Boolean queries can be used to determine joint bucket membership without leaking the bucket membership of individual attributes.

In our example, the first four rows belong to the same bucket combination $(\text{Gen}_1, \text{Dept}_1)$, while the last row with ID 5 belongs to bucket combination $(\text{Gen}_1, \text{Dept}_2)$. The server determines the (encrypted) shift for each row by evaluating the polynomial[2] P on the encrypted grouping monomials $P(G_1, G_2) = a_0 + a_1 \cdot G_1 + a_2 \cdot G_2 + a_3 \cdot G_1 \cdot G_2$. For the first row, the

[2]This is possible without calling $\otimes$ since the polynomial's coefficients are outsourced in plaintext.

1500	5000	0	4000

(a) Bucket combination $(\text{Gen}_1, \text{Dept}_1)$

0	0	0	2000

(b) Bucket combination $(\text{Gen}_1, \text{Dept}_2)$

Figure 2.4: Sketch of decrypted aggregation result for different buckets.

polynomial evaluates to 1, for the second row to 2^{64} and so on. Notice that the result of the evaluation is encrypted, hence hiding the shift value in each bucket. Multiplication of the corresponding entry in the value column Salary by the encrypted shift yields the shifted value to be aggregated. The server adds up all the shifted values for each bucket combination separately and returns the result. By decrypting the result, the client receives the packed plaintexts depicted in Figure 2.4. Since the client has chosen the shifts of all value combinations by determining the polynomial's coefficients, the client can map each part of the plaintext to a group attribute combination. Table 2.8 shows the final result of the query. Note that the coefficients can be reused for other attributes if they use the same bucket size.

SUM(Salary)	Gender	Department
4000	male	Sales
5000	female	Sales
1500	female	Finance
2000	male	Facility

Table 2.8: Result of the query stated in Listing 2.2 on Table 2.2.

2.6.3 Formalization

Before we give a comprehensive formal description, we define the interface of the SAGMA construction for secure aggregation of k aggregation values and supporting a combination of up to t arbitrary grouping attributes in one query. SAGMA consists of the following six probabilistic polynomial time (PPT) algorithms. We refer to Table 2.9 for an overview of the used variables in our algorithm descriptions.

$(pp, K) \leftarrow \text{Setup}(1^\lambda, D_1, \ldots, D_l)$: Executed on the client. Generates the cryptographic keys and outputs the public parameters pp and the secret key K. $D_1, \ldots, D_l$ denote the value domains of the grouping attributes.

$C \leftarrow \text{EncTable}(K, \{\{v_{i,j}\}_{j=1}^{k}, \{g_{i,j}\}_{j=1}^{l}\}_{i=1}^{rows})$: Executed on the client. Using the given cryptographic keys, it encrypts the table executing EncRow for each row. Depending on the concrete construction, an index for grouping or additional filtering is created. The result is then outsourced to the untrusted server.

$c \leftarrow \text{EncRow}(pk, v_1, \ldots, v_k, g_1, \ldots, g_l)$: Executed on the client for each row. Encrypts the database row. This operation also allows the client to add subsequent rows to the database after initial encryption.

$t_{grp} \leftarrow \text{AggGrpByToken}(K, V, Q)$: Executed on the client. Creates a grouping token to execute grouping by up to t grouping columns in Q and aggregation of value columns in V.

$c_{agg} \leftarrow \text{AggGrpBy}(pp, t_{grp}, C)$: Executed on the server. Aggregates the encrypted data with GroupBy statement using the tokens generated in the previous step.

$\text{res} \leftarrow \text{DecAgg}(K, c_{agg})$: Executed on the client. Decrypts the encrypted result of the aggregation query with GroupBy statement.

Variable	Description
D_i	Value domain for i-th grouping attribute
l	Number of grouping attributes
λ	Security parameter
f_i	Pseudorandom function mapping grouping attribute values of column i to natural numbers
a_i	Polynomial coefficients used for oblivious shift calculation
B	Bucket size of each bucket containing B group attribute values
pp	Public parameters
K_{SSE}	Secret key for searchable encryption scheme
K	Secret master key
$\|S\|$	Number of elements in set S
$v_{i,j}$	Attribute value for the j-th value column (in the i-th row if stated)
$g_{i,j}$	Attribute value for the j-th grouping column (in the i-th row if stated)
k	Number of value attributes
D	Document collection for SSE containing the bucket identifiers of all rows
I	Searchable encrypted index for bucket memberships of all grouping attribute values for each row
$r_{i,j}$	Offset of value in bucket for j-th grouping column in i-th row
C	Encrypted database
m_i	Monomial for oblivious shift calculation of bucketized group values
Q	GroupBy clause of aggregation query for up to t grouping attributes
s_i	Number of buckets for i-th grouping attribute
$t_{i,j}$	SSE token for the j-th bucket of the i-th grouping attribute
p	Size of joint bucket
R	Rows for aggregation of joint bucket
S_i	Shift for the i-th row

Table 2.9: Overview of used variables.

Based on the formalization, we discuss storage efficiency in Section 2.6.4, state a theoretical upper bound for the information leakage of our scheme in Section 2.6.4, and compare our construction with an approach based on pre-computation and Seabed in Section 5.2.3.

Let $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec}, \oplus, \otimes)$ be a semantically secure SWHE scheme supporting one single multiplication as introduced before. Let $SSE = (\text{Gen}_{SSE}, \text{Enc}_{SSE}, \text{Token}_{SSE}, \text{Search}_{SSE})$ be an adaptively semantically secure SSE scheme as defined in Section 2.3.3. In the following, we give an intuition for each SAGMA algorithm together with a formal description in Algorithms 1– 6.

Algorithm 1 Key generation algorithm.

Setup$(1^{\lambda_0}, 1^{\lambda_1}, D_1, \ldots, D_l)$:
$(pk, sk) \leftarrow \text{Gen}(1^{\lambda_0})$
$F = (f_1, \ldots, f_l),\ f_i : D_i \mapsto \{0, \ldots, |D_i|\}$
$pp = (pk, a_0, \ldots, a_{B^l-1})$
$K_{SSE} \leftarrow \text{Gen}_{SSE}(1^{\lambda_1})$
return $pp, K = (pk, sk, F, K_{SSE})$

Setup as stated in Algorithm 1 generates the cryptographic keys, defines the mapping functions for the group domains and calculates the polynomial determining the shift values.

EncTable as stated in Algorithm 2 creates and indexes the group buckets for all grouping attributes using SSE. Note that EncRow is called for a set of rows encrypting aggregation values and bucketized group values.

Algorithm 2 Encryption algorithm for the complete table.

EncTable$(K, \{\{v_{i,j}\}_{j=1}^{k}, \{g_{i,j}\}_{j=1}^{l}\}_{i=1}^{rows})$:
$D = \{\{j : \frac{f_j(g_{i,j})}{B}\}_{j=1}^{l}\}_{i=1}^{rows}$
$I \leftarrow \text{Enc}_{SSE}(K_{SSE}, D)$
$r_{i,j} = f_j(g_{i,j}) \bmod B$
$c_i \leftarrow \text{EncRow}(pk, v_{i,1}, \ldots, v_{i,k}, r_{i,1}, \ldots, r_{i,l})$
return $I, C = c_1, \ldots, c_{rows}$

Particularly, *EncRow* as stated in Algorithm 3 determines monomials of the bucketized group values required to determine appropriate shifts on the server later on and applies somewhat homomorphic encryption to the aggregation values and the monomials to support dynamic shifting. Note that this algorithm can be used for database updates after the initial table encryption if the bucket index I is updated accordingly.

Algorithm 3 Encryption algorithm for a single row.

EncRow$(pk, v_1, \ldots, v_k, g_1, \ldots, g_l)$:

$$m_1, \ldots, m_{B^l} = \left\{ \prod_{i=1}^{l} g_i^{e_i} \mid \forall\ 0 \leq e_i < B \right\}$$

return $c = (\llbracket v_1 \rrbracket_{\oplus}^{\otimes}, \ldots, \llbracket v_k \rrbracket_{\oplus}^{\otimes}, \llbracket m_1 \rrbracket_{\oplus}^{\otimes}, \ldots, \llbracket m_{B^l} \rrbracket_{\oplus}^{\otimes})$

AggGrpByToken as stated in Algorithm 4 creates SSE search tokens for all buckets of the grouping attributes and outputs the identifier of the attribute to aggregate, the identifiers of the group attributes, and the SSE tokens.

Algorithm 4 Group token generation algorithm.

AggGrpByToken$(K, V \in \{1, \ldots, k\}, Q \subseteq \{1, \ldots, l\})$:

for all $q \in Q$ **do**

$s_q = \left\lceil \frac{|D_q|}{B} \right\rceil$

for all $1 \leq b \leq s_q$ **do**

$t_{q,b} \leftarrow \text{Token}_{SSE}(K_{SSE}, q : b)$

return $V, Q, \{t_{q,b} \mid q \in Q, b \in \{1, \ldots, s_q\}\}$

AggGrpBy as stated in Algorithm 5 uses the encrypted index to determine the rows of all buckets of the group attributes. By calculating the intersection, rows belonging to joint buckets are determined and aggregation is executed for each joint bucket. Aggregation involves determining the appropriate shift for each row and multiplying the shift by the value attribute. Finally, one encrypted result is returned for each joint bucket.

DecAgg as stated in Algorithm 6 decrypts and unpacks the packed ciphertexts that result from aggregation.

2.6.4 Efficiency

By precomputing $B^l - 1$ monomials for the polynomial evaluation, a SWHE scheme supporting one multiplication is sufficient for our construction. However, the bucket size B and the number of group attributes l is limited by the available storage space on the server. Particularly, our SAGMA construction requires $B^l - 1$ monomials with exponential increase in the number of group attributes l.

Generally, only small subsets of group attributes occur together in one query. Limiting the number of group attributes stated in one query allows us to reduce the number of required monomials.

[3] Enumerates joint buckets using the cartesian product of the buckets of the queried attributes.

Algorithm 5 Aggregation algorithm over encrypted data.

AggGrpBy$(pp, V, Q, \{t_{q,1}, \ldots, t_{q,s_q} \mid q \in Q\}, C, I)$:

$s_q = \left\lceil \frac{|D_q|}{B} \right\rceil$, for $q \in Q$

$p = B^{|Q|} - 1$

determine coefficients $a_0, \ldots, a_p$ for Q

determine indices $i_1, \ldots, i_p$ for Q

for all $(b_1, \ldots, b_{|Q|}), b_i \in \{1, \ldots, s_{Q_i}\}$ [3] **do**

$R \leftarrow \bigcap_{i=1}^{|Q|} \mathrm{Search}_{SSE}(I, t_{Q_i,b_i})$

for all $r \in R$ **do**

$S_r = [\![a_0]\!]_{\oplus}^{\otimes} \oplus \bigoplus_{j=1}^{p} a_j \otimes [\![m_{i_j}]\!]_{\oplus}^{\otimes}$

$agg_{b_1,\ldots,b_{|Q|}} = \bigoplus_{r \in R} [\![v_V]\!]_{\oplus}^{\otimes} \otimes S_r$

return all aggregates $agg_{b_1,\ldots,b_{|Q|}}$

Algorithm 6 Decryption algorithm for query result.

DecAgg$(K, agg_1, \ldots, agg_s)$:

for $1 \leq i \leq s$ **do**

$u_{i,1}, \ldots, u_{i,B^{|Q|}} \leftarrow \mathrm{Dec}(sk, agg_i)$

return $u_{1,1}, \ldots, u_{s,B^{|Q|}}$

More particular, assume the number of attributes is limited by a constant t. We denote the number of monomials that have to be stored for each row in a database with l group attributes by $m(l,t)$. We can apply our naïve construction to all subsets of attributes of size t, which requires to store $m(l,t)_{\text{naive}}$ monomials per row as defined by the following equation:

$$m(l,t)_{\text{naive}} = \binom{l}{t} \cdot (B^t - 1) \leq \left(\frac{l \cdot e}{t}\right)^t \cdot (B^t - 1)$$

As a result, the number of required monomials is polynomially bounded, instead of exponentially in l. However, the monomial reuse in our improved scheme reduces the number of required monomials further. To calculate the exact number of monomials necessary, we examine how many monomials are necessary to support grouping of t attributes, if we already support grouping of $t-1$ attributes. Notice that $m(l,0) = 0$. The result of this iterative construction is given for some t in Table 2.10. In the first step, we aim to support grouping of a single attribute. This requires storing $B-1$ powers of all attributes. To support grouping of two attributes $B^2 - 1$ monomials are required for each subset of attributes of size two (-1 because of the empty

$\mathbf{t}$	$\mathbf{m(l,t) - m(l,t-1)}$
1	$l \cdot (B-1)$
2	$\binom{l}{2} \cdot \left(B^2 - 1 - 2 \cdot (B-1)\right)$
3	$\binom{l}{3} \cdot \left(B^3 - 1 - \binom{3}{1} \cdot (B-1) - \binom{3}{2} \cdot (B-1)^2\right)$
$\vdots$	$\vdots$
t	$\binom{l}{t} \cdot \left(B^t - \sum_{i=0}^{t-1} \binom{t}{i} (B-1)^i\right)$

Table 2.10: Required number of monomials to support grouping up to t attributes.

product). The $B-1$ powers of the two attributes can be used and thus be subtracted. This can be generalized for an arbitrary t resulting in the following equation:

$$\begin{aligned} m(l,t) - m(l,t-1) &= \binom{l}{t} \cdot \left(B^t - \sum_{i=0}^{t-1} \binom{t}{i} (B-1)^i\right) \\ &\overset{*}{=} \binom{l}{t} \cdot (B-1)^t . \end{aligned}$$

The above transformation $(*)$ can be verified using a proof by induction. More precisely, we define the number of monomials required to support grouping of t attributes in one query given a database with l possible group attributes as:

$$m(l,t) = \sum_{i=1}^{t} m(l,i) - m(l,i-1) = \sum_{i=1}^{t} \binom{l}{i} \cdot (B-1)^i.$$

A lower bound of this number grows polynomially in l and thus $m(l,t) \in \Theta(l^t \cdot B^t)$:

$$\sum_{i=1}^{t} \binom{l}{i} \cdot (B-1)^i \geq \binom{l}{t} \cdot (B-1)^t \geq \left(\frac{l}{t}\right)^t \cdot (B-1)^t.$$

Security

Given a semantically secure SWHE encryption scheme Σ and an adaptively semantically secure SSE scheme, the construction as stated in Algorithms 1–6 essentially leaks the bucket membership of all queried group attributes. We emphasize that our construction offers improved security compared to the encryption of individual group attributes either deterministically or searchable. While deterministic encryption as well as searchable encryption leaks the frequencies of the whole plaintext domain after the execution of one aggregation query, our construction only leaks the frequencies of distinct buckets.

For a formal security analysis, we use a simulation-based security definition following the work on searchable symmetric encryption by Curtmola et al. [47]. Basically, simulation-based security definitions consist of two experiments, the real and the simulated experiment. The real experiment describes a regular protocol sequence, i.e., in our case an adversary $\mathcal{A}$ chooses a plaintext database table to be encrypted and queries the grouping tokens to be created. Encryption of the database table and generation of grouping tokens is executed by the regular algorithms of the scheme. In contrast, in the simulated experiment the encrypted database and the grouping tokens are created by a simulator $\mathcal{S}$ that only has access to limited information about the plaintext database and the queries. This limited information is modeled by a leakage function $\mathcal{L}$, whose output is given to the simulator. Both experiments output the encrypted database and the created grouping tokens. Intuitively, a scheme is secure if the output of both experiments is computationally indistinguishable, i.e., no PPT algorithm exists that has non-negligible advantage in distinguishing the two distributions. A formal description of these two experiments is given in Figure 2.5. The security notion is stated in Definition 4. Notice, that our security definition is adaptive, i.e., the adversary has access to the grouping tokens created for earlier queries and can choose the next query in dependence on the tokens. In contrast, non-adaptive security definitions require the adversary to choose all queries at once, which is a weaker security definition and a far less realistic scenario for applications.

Definition 4 (Adaptive $\mathcal{L}$-Security)**.** *The SAGMA scheme as defined in Section 2.6.3, is adaptively $\mathcal{L}$-secure, if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_{q+1})$, there exists a PPT simulator $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_{q+1})$ such that for all PPT algorithms $\mathcal{D}$ it holds that*

$$\begin{aligned}&\big|\Pr\left[\mathcal{D}(v, st_{\mathcal{A}}) = 1 : (v, st_{\mathcal{A}}) \leftarrow \mathbf{Real}^*{}_{\mathcal{A}}(\lambda)\right] \\ &\quad - \Pr\left[\mathcal{D}(v, st_{\mathcal{A}}) = 1 : (v, st_{\mathcal{A}}) \leftarrow \mathbf{Sim}^*{}_{\mathcal{A},\mathcal{S}}(\lambda)\right]\big| \leq \mathrm{negl}(\lambda)\end{aligned}$$

where $q = \mathrm{poly}(\lambda)$ and $\mathrm{negl}(\lambda)$ is negligible in λ.

Formalized Information Leakage We construct a simulator $\mathcal{S}$ fulfilling Definition 4. $\mathcal{S}$ has restricted information in form of the identifiers of the value attribute and the group attributes (V, Q) of all queries. Both V and Q are only identifiers of the queried attributes and do not contain the actual values of these attributes, i.e., they identify the column to be aggregated and to be grouped by. Further, the trace τ, that is the information leaked by SSE is forwarded to $\mathcal{S}$, hence the overall leakage we proof as upper bound is formalized as:

$$\mathcal{L}\left(T, (V_1, Q_1) \ldots, (V_i, Q_i)\right) = \left((V_1, Q_1), \ldots, (V_i, Q_i), \tau_i\right).$$

We do not explicitly mention the number of rows and columns of the database and the bucket size B in the leakage but assume them to be public. Notice that each aggregation query involves multiple SSE queries, one for each bucket of the queried group attributes. Thus, the trace after i aggregation queries contains the access and the search pattern of $i' \geq i$ keyword queries, namely $\tau_i = \tau(D, w_1, \ldots, w_{i'})$. The trace itself consists of the sizes of the documents and the access pattern (i.e. the document identifiers of matching documents) and the search pattern that discloses if two tokens correspond to the same keyword. In our case, the search pattern corresponds to a bucket identifier and the access pattern reveals the rows contained in each bucket. In summary, the overall leakage of our SAGMA construction can be described as follows: the identifiers of the grouping attributes are leaked and for each queried group attribute, the construction leaks the mapping of rows to bucket identifiers, which is included in the access pattern of SSE.

We use SSE as black box, including its common security definition as introduced by Curtmola et al. [47].

Real$^*_{\mathcal{A}}(\lambda)$:
$D_1, \ldots, D_l, st_{\mathcal{A}} \leftarrow \mathcal{A}_0(1^\lambda)$
$pp, K \leftarrow \text{Setup}(1^\lambda, D_1, \ldots, D_l)$
$T, st_{\mathcal{A}} \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, pp)$
$C \leftarrow \text{EncTable}(K, T)$
for $1 \leq i \leq q$ **do**
$Q_i, st_{\mathcal{A}} \leftarrow \mathcal{A}_{i+1}(st_{\mathcal{A}}, pp, C, t_{grp,1}, \ldots, t_{grp,i-1})$
$t_{grp,i} \leftarrow \text{AggGrpByToken}(K, (V_i, Q_i))$
return $pp, C, t_{grp,1}, \ldots, t_{grp,q}, st_{\mathcal{A}}$

Sim$^*_{\mathcal{A},\mathcal{S}}(\lambda)$:
$D_1, \ldots, D_l, st_{\mathcal{A}} \leftarrow \mathcal{A}_0(1^\lambda)$
$pp, st_{\mathcal{S}} \leftarrow \mathcal{S}_0(1^\lambda, D_1, \ldots, D_l)$
$T, st_{\mathcal{A}} \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, pp)$
$C, st_{\mathcal{S}} \leftarrow \mathcal{S}_1(st_{\mathcal{S}}, \mathcal{L}(T))$
for $1 \leq i \leq q$ **do**
$Q_i, st_{\mathcal{A}} \leftarrow \mathcal{A}_{i+1}(st_{\mathcal{A}}, pp, C, t_{grp,1}, \ldots, t_{grp,i-1})$
$t_{grp,i}, st_{\mathcal{S}} \leftarrow \mathcal{S}_i(st_{\mathcal{S}}, \mathcal{L}(T, (V_1, Q_1), \ldots, V_i, Q_i))$
return $pp, C, t_{grp,1}, \ldots, t_{grp,q}, st_{\mathcal{A}}$

Figure 2.5: Security Experiments – Real vs. Ideal

Theorem 1. *If the used SSE scheme is adaptively semantically secure, and the used SWHE scheme Σ is semantically secure, then our SAGMA construction as stated in Algorithms 1– 6 is adaptively semantically $\mathcal{L}$-secure according to Definition 4.*

Proof. For our proof sketch we give an intuition why a simulator exists such that the outputs of the experiments $\mathbf{Real}^*_{\mathcal{A}}$ and $\mathbf{Sim}^*_{\mathcal{A},\mathcal{S}}$ are computationally indistinguishable for every possible PPT adversary $\mathcal{A}$. Using a standard hybrid argument, we show that our construction satisfies the security definition. The output of both experiments has the same structure, that is, $pp, C' = (C, I), t_{grp,1}, \ldots, t_{grp,q}, st_{\mathcal{A}}$. Since the public parameters pp have been created by the same algorithm Setup, their distribution is identical. If a PPT algorithm exists that distinguishes the encrypted database C of the two experiments, then an adversary can be constructed breaking semantic security of Σ contradicting our assumption. Similarly, if the encrypted index I or the SSE tokens contained in the grouping tokens t_{grp} can be distinguished, then the security of the SSE scheme can be broken. Furthermore, the grouping tokens t_{grp} contain the identifiers of the queried value attribute and group attributes. Since these identifiers are contained in the leakage, they are identical in both experiments.

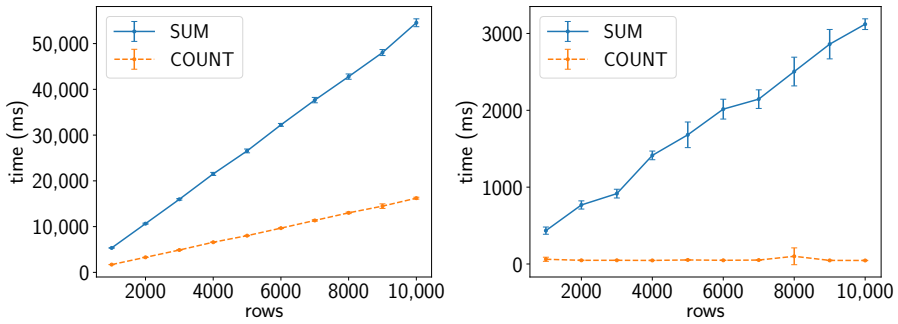
Because no component of the output can be distinguished by a PPT algorithm, using the hybrid argument it follows that the outputs of both experiments are indistinguishable and thus our construction fulfills the security definition. □

2.6.5 Evaluation

We implemented SAGMA described in Section 2.6.2 using the SWHE scheme published by Boneh, Goh and Nissim (BGN) [21] based on bilinear maps. BGN is additively homomorphic and supports a single multiplication of ciphertexts. We implemented SAGMA in Java using parallelization during query execution and decryption to use multiple cores.

Decryption of BGN requires to calculate the discrete logarithm in a prime-order group; thus, the plaintext space has to be restricted to support efficient decryption. Nevertheless, the decryption of a ciphertext with restricted plaintext space of 32 bits still takes several seconds on a current laptop and is too inefficient for data aggregation. Hu et al. propose to use the Chinese remainder theorem (CRT) to speed up the decryption of BGN [101]. They split a message into several parts; homomorphic operations are executed for each part and the result is reconstructed after the decryption of all parts using CRT. This approach offers a trade-off between decryption time on the client and execution time of homomorphic operations on the server. Since ciphertext components can only be decrypted if they are small enough and homomorphic operations increase the components, the use of homomorphic operations is limited.

Our implementation supports three aggregation operations, namely summation, row count and average. The row count can be calculated by aggregating the shifts instead of the shifted values. As the result is limited by the total number of rows, the CRT scheme is not required.



(a) Aggregation time (b) Decryption time

Figure 2.6: Processing time of our scheme SAGMA for a varying number of rows.

Results

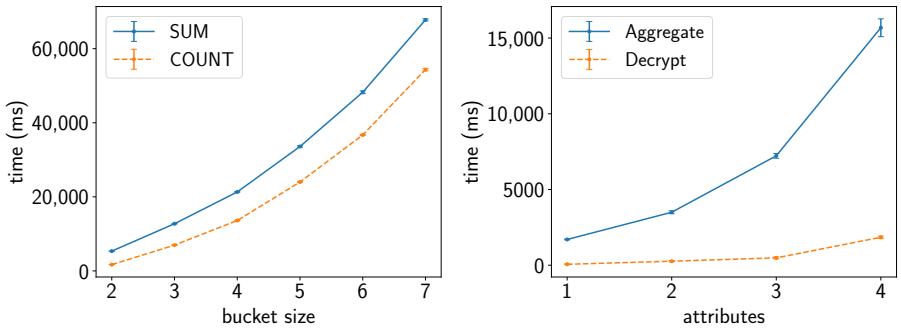
We use the *lineitem* table of the TPC-H [4] benchmark to evaluate aggregation time and decryption time. The evaluation runs on a machine with two Intel Xeon E5-2670 CPUs with eight cores each operating at 2.60 GHz and 256 GB of RAM. We execute every query ten times and calculate the mean running time, as well as the 95% confidence interval. Network latencies between client and server are not considered. Since the performance of SSE schemes has been analyzed extensively before, our implementation uses a plaintext index located on the client to determine the bucket identifiers of rows. For our evaluation, we instantiate a cryptographic key with 1024 bits. Since BGN is based on the hardness assumption of factorization of a composite modulus this choice provides about 80 bits of security [10, 72].

We evaluate the performance of our SAGMA construction in dependence on the number of rows to be aggregated. Figure 2.6 shows a linear increase of the aggregation time. Aggregating the count of 1000 rows requires 1.9 s and for $10,000$ rows it lasts 17.7 s. Summation is less efficient, since it is based on the CRT scheme. The decryption time of the count operation stays constant, while it increases for the summation operation. Again, this is due to the CRT construction limiting the number of supported additions.

We emphasize that most queries contain `WHERE` clauses to filter rows which limits the number of rows that have to be aggregated in real use cases making our construction also suitable for larger databases. Such preceding selection is orthogonal to our work and can be implemented efficiently using SSE.

The runtime for different bucket sizes B is compared in Figure 2.7a. The aggregation time increases superlinearly, summation takes 5.7 s for bucket size 2 and 71.3 s for bucket size 7 due to the use of unit shifts to reduce

[4] See `https://www.tpc.org/tpch/`.



(a) Varying bucket size (b) Varying attribute number

Figure 2.7: Aggregation time of our scheme SAGMA for varying bucket sizes and a varying number of combined attributes in one query.

the size of the CRT components. Instead of one polynomial of degree B, B polynomials are required to evaluate the shifts for each row. Hence, the running time increases quadratically in the bucket size. Again, the row count operation is more efficient than summation.

Finally, we evaluate the impact of the number of attributes to be grouped. According to Figure 2.7b, the aggregation time increases superlinearly. Here, the running time increases polynomially in the total number of attributes of the database table if the number of grouping attributes in a single query is limited by t. This enables us to store and combine B^t monomials to evaluate the polynomial.

Chapter 3

Electronic Payment

The contents of this chapter are based on joint work with Dirk Achenbach, Roland Gröll, Alexander Koch, Bernhard Löwe, Jeremias Mechler, Jörn Müller-Quade, Jochen Rill, and Julian Herr. Parts of the content previously appeared in other publications. The publications are listed below. This thesis presents a new proof of security based on the computer-assisted protocol verification tool Tamarin.

- Dirk Achenbach, Roland Gröll, Timon Hackenjos, Alexander Koch, Bernhard Löwe, Jeremias Mechler, Jörn Müller-Quade, and Jochen Rill. "Your money or your life—modeling and analyzing the security of electronic payment in the UC framework". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2019, pp. 243–261. DOI: 10.1007/978-3-030-32101-7_16 [1]
- Jochen Rill. "Towards Applying Cryptographic Security Models to Real-World Systems". PhD thesis. Karlsruhe Institute of Technology (KIT), 2020 [144]
- Alexander Koch. "Cryptographic protocols from physical assumptions". PhD thesis. Karlsruhe Institute of Technology (KIT), 2019 [113]

3.1 Introduction

"Your money, or your life!"—surrender your belongings or face death. This threat was used by bandits in England until the 19th century [132]. As people often needed to carry all their valuables with them when traveling, banditry was a lucrative (albeit dangerous) endeavor. Today, electronic money transfer (EMT) systems alleviate the need to have one's valuables at hand but introduce new threats as well. Instead of resorting to violence, modern thieves may compromise their victim's bank account. Once they are widely deployed, insecure EMT systems are notoriously difficult to transition

away from—magnetic stripes are still in use today. The current state-of-the-art payment standard EMV (short for *Europay International, MasterCard and VISA*, also known as "Chip and PIN") improves on this but falls short of providing a secure solution to payment (or money withdrawal), as shown by its many weaknesses described in literature. Among these are practical attacks, such as

(i) "cloning" chip cards by pre-computing transaction messages (so-called "pre-play attacks") [20],

(ii) tricking an innocent customer into accepting fraudulent transactions by relaying transaction data from a different point of sale (POS) (so-called "relay attacks") [57],

(iii) disabling the Personal Identification Number (PIN) verification of stolen cards by intercepting the communication between chip card and POS device [129, 13].

Upon close examination of these attacks, one finds that these issues mainly stem from *two major false assumptions* which are baked into the design of the EMV protocol:

(i) that the input device (POS or automated teller machine (ATM)) itself is trustworthy and

(ii) that the communication between all protocol participants (e.g. between the chip card and the POS) cannot be intercepted.

Designing payment protocols that are secure even if the input device is infected with malware is especially challenging, since the input device provides the user with an interface to communicate with the bank. A human user is not capable of performing cryptographic operations and thus needs such a device. However, interacting with a compromised input device, the user cannot be sure that the displayed transaction is really executed and not manipulated behind the scenes.

Even though these assumptions are critical for the security of EMV, they are not explicitly stated in the standardization documents [61, 62, 63]. We suggest that this is mainly because EMV has been created by a functionality-focused engineering process in which problems are fixed as they occur and features are added when necessary, rather than a design process that uses formal models and techniques.

Modern cryptographic protocols in contrast are designed by first providing a formal description of the protocol, explicitly stating all necessary assumptions and then giving a proof of security. This does not make cryptographic protocols unbreakable, but it does make their potential breaking points explicit. Therefore, we argue that it is necessary to start developing electronic payment protocols by using the same methodology of rigorous formal modeling as has already been established in cryptography.

3.2 Secure Electronic Payment

As a basis for our considerations, observe the process of withdrawing cash at an automated teller machine (ATM). First, there is the bank and its customer, Alice. Second, there is the money dispensing unit inside the ATM. Assuming authenticated communication from Alice to the bank and from the bank to the money dispensing unit, secure payment is easy: Alice communicates the amount of cash she needs and the identity of the money dispensing unit she expects to receive the cash from. The bank then instructs the money dispensing unit to dispense the money. However, Alice is a human and therefore cannot perform cryptographic operations required for a classical channel establishment protocol. Thus, Alice needs another party which offers a user interface to her and communicates with the bank, namely an ATM.

This does not only apply to cash withdrawal but can be extended to electronic money transfer (EMT) in general. To this end, think of Alice as the *initiator* of a transaction and the money dispensing unit as the *receiver*. The process of money withdrawal can now be framed as a payment of money from Alice's account to the account of the money dispensing unit, which upon receiving money, promptly outputs cash. In this scenario the ATM serves as an *input device*. The same works for the point of sale: here, the device's owner (e.g. the supermarket) is the receiver.

Regarding our adversarial model, we make no assumption about the trustworthiness of the ATM whatsoever and do assume that the adversary has control over all communication. However, we assume the money dispensing unit inside the ATM (or receiver in general) to be trusted. If it is under adversarial control, the adversary could simply dispense money at will. This requires that the money dispensing unit can only be controlled remotely by the bank and not by the ATM itself.

3.2.1 Necessary and Sufficient Requirement

The core challenge when realizing secure electronic payment is the authenticated transmission of transaction data from the (human) initiator to the bank. As discussed before, we assume that the bank can communicate with the (trusted) receiver authentically. This can be realized with standard cryptographic mechanisms such as digital signatures as the sender and the receiver are both not human. Thus, in this work we focus on the secure initiation of a transaction by a human.

In an ideal payment protocol, the (human) initiator would transmit the desired transaction data securely to the bank without the adversary being able to interfere. Thus, by encoding a message inside the transaction data, a payment protocol can be used to transmit a message authentically. We use this insight to establish a *necessary* condition for protocols that realize secure electronic payment: they must be strong enough to realize authenticated

communication between the initiator and the bank. Protocol designers can use this condition as an easily checkable criterion for the *insecurity* of payment protocols. Likewise, authenticated communication between the initiator and the bank is *sufficient* to realize electronic payment, as the authenticated channel can be used to transmit the transaction details authentically.

3.2.2 Confirmation Is Key

Realizing authenticated communication from a human initiator to the bank is hard since a human cannot execute cryptographic operations. Furthermore, the ATM that provides the user with an interface to communicate with the bank might be compromised. Since the human initiator of a transaction cannot be sure that an untrusted input device correctly processes his transaction data, he needs a way of confirming the transaction data with the bank before the transaction is processed. By default, EMV uses smartcards containing shared secrets with the bank in order to authenticate transactions. However, this only works if the input device which accesses the smartcard (e.g. the ATM) can be trusted. Otherwise, after the initiator enters his Personal Identification Number (PIN) to authorize a seemingly legitimate transaction, the input device can present false (transaction) data to the smartcard (cf. e.g. [20]). Thus, smartcards are not sufficient to establish a secure confirmation mechanism. Instead, we need a mechanism that is transparent for the user even if the ATM is compromised. This can for example be realized using an additional device with a display. Similar to a smartcard, the additional device might store a secret key. However, the major advantage is that the device can show the transaction data before signing it and request confirmation from the user, e.g., by pressing a button.

3.2.3 One-out-of-two Security

Various devices such as transaction authentication number (TAN) generators and smartphones can be used to establish a confirmation mechanism. However, smartphones, which are increasingly used to replace smartcards, regularly call attention because of vulnerabilities. They are complex systems connected to the Internet and are thus vulnerable to attacks—especially if they are operated by people without expertise in IT security. As stated earlier, ATMs are vulnerable to attacks too. Unpatched operating systems and exposed Universal Serial Bus (USB) interfaces are only two examples for weaknesses that have been exploited successfully in the past to infect ATMs.

This dilemma can be resolved by requiring trust in only *one of the two devices*. We call this property *one-out-of-two security*. This means that a protocol is still secure if one of the two devices is corrupted, no matter which one of them. We propose *one-out-of-two security* as a means of providing protection against malware attacks.

We argue that it is difficult for an adversary to compromise both an ATM and the smartphone of a user that wants to withdraw cash at an ATM. Especially since the two devices differ in many aspects such as their location, hardware, operating system and system configuration. Thus, it is unlikely that an adversary is able to compromise both devices using the same method. Therefore, the required effort for a successful attack is heavily increased.

3.3 On the Security of Current Payment Protocols

In this section, we analyze current protocols for withdrawing cash and paying at the point of sale (POS). The protocols discussed in this section make additional implicit assumptions, which we believe to be plausible, but want to make explicit. These include the following:

(i) An additional trusted device beside the input device. This is a plausible assumption if the device is simple, less so if it is a smartphone. However, using an additional device could enable protocols to provide one-out-of-two security.

(ii) Authenticated communication between the initiator of a transaction and an additional trusted device. This is a realistic assumption, since the initiator owns the device. Likewise, the initiator can authenticate themselves to the device, e.g., by unlocking the screen of a mobile device.

(iii) Confidential communication from the initiator to the ATM, which can be realized by covering the PIN pad with one's hand if the ATM is not compromised.

(iv) Confidential communication from the ATM to the bank. This can be realized using public-key cryptography.

The default EMV protocol with smartcard and PIN does not rely on an additional device and thus assumption (i) and (ii) are not required for this protocol. In the following, we analyze multiple protocols for cash withdrawal and paying at the POS. Table 3.1 summarizes our findings.

3.3.1 EMV

Even though EMV is the most widely used standard for payments, we do not elaborate on its security in this chapter. As mentioned before, its design incorporates at least two assumptions that do not hold, as several attacks have been demonstrated. Current payment protocols such as Google Pay, Apple Pay, Samsung Pay, Microsoft Pay and Garmin Pay provide an app that uses the EMV contactless standard to communicate with existing POS devices via near-field communication (NFC) [151, 158]. Since they rely on

Table 3.1: Comparison of current payment protocols. A protocol is marked as offline, if the additional device does not require an Internet connection during the payment process or no additional device is used.

Protocol	Offline	Secure	Applicable for
EMV standard	✓	×	Withdrawal, PoS
Cardless Cash	✓	×	Withdrawal
VR-mobileCash	×	×	Withdrawal
L-Pay (our protocol)	✓	✓: 1-of-2	Withdrawal, PoS

Consumer Device Cardholder Verification Method (CDCVM), the user is authenticated by the mobile device exclusively. Currently, these apps use a PIN, a fingerprint or face recognition and thus do not incorporate a second device such as the POS device for authentication. Therefore, the security of the protocol is solely based on the mobile device.

3.3.2 Cardless Cash

Cardless Cash [42] is an app-based protocol for cash withdrawal offered by numerous banks in Australia. In its most simple variant, it works as follows: after registration, the app can be used to create a "cash code" by entering the desired amount and a phone number. The phone number is used to send a PIN via SMS. To dispense the cash, the PIN has to be entered at the ATM alongside the cash code. The security of the protocol is based on both the security of the ATM and the mobile device. An attacker that controls the mobile device can generate arbitrary cash codes and receive the PINs via SMS. Similarly, since all relevant information is entered on the ATM, an attacker controlling the ATM can eavesdrop on the cash code and PIN. Instead of sending this data to the bank, the attacker can use it to withdraw the money at another ATM. Thus, if any of the two devices is compromised, the protocol is insecure.

3.3.3 VR-mobileCash

VR-mobileCash [141] is another app-based protocol for cash withdrawal offered by Volks- und Raiffeisenbanken, a German association of banks. Upon registration, the user receives the mobile personal identification number (mPIN), which has to be entered on the ATM later on to confirm a transaction. To withdraw cash, the user has to enter the desired amount in the app. After selecting the mobile payment option at the ATM, the ATM shows a mobile transaction identification number (mTIN) which has to be entered in the app. The ATM then shows the requested amount and asks the user to enter the mPIN. If the mPIN is correct, the ATM dispenses the requested amount of cash.

Although not stated explicitly in the public documentation, the mobile device has to be online during the transaction, as the ATM is informed about the transaction data without establishing a communication mechanism with the ATM locally. In its current form, VR-mobileCash does not fulfill *one-out-of-two security*. If the mobile device is corrupted but the ATM is honest, a user can detect an attack because he has to confirm the transaction by entering the mPIN at the ATM and thereby verifies the location of the ATM. However, a *corrupted ATM* can employ a relay attack by displaying the mTIN of another corrupted ATM and forwarding the entered mPIN to it, thus allowing the second corrupted ATM to dispense the cash. This could be fixed by adding a serial number imprinted on the ATM which is also displayed in the app after entering the mTIN. Thereby VR-mobileCash could potentially realize *one-out-of-two security*.

3.3.4 Online Banking

Some protocols that are currently used in online banking such as the chip authentication program (CAP) protocol and photoTAN establish a confirmation mechanism using an additional device such as a TAN generator or smartphone. However, they are not intended to be used for cash withdrawal or paying at the POS. We analyze the security of online banking schemes in more detail in Chapter 4.

3.4 Designing A Protocol for Electronic Payment

Even though many of the current payment protocols rely on smartphone apps to secure the payment process, none of them provides *one-out-of-two security*. Thus, compromising an automated teller machine (ATM) is still a feasible way to steal money. We design a protocol for electronic payment and aim to achieve the following design goals:

(i) Establish a confirmation mechanism using a smartphone

(ii) Achieve one-out-of-two security

(iii) Do not require internet connectivity for the smartphone

Our protocol L-Pay is inspired by online banking protocols such as chip authentication program (CAP) and photoTAN. We establish a confirmation mechanism using the initiator's smartphone. In addition, we require the initiator to confirm a transaction by entering a Personal Identification Number (PIN) at the ATM. This prevents a compromised smartphone from initiating arbitrary transactions. In combination, L-Pay achieves *one-out-of-two security*. Before transactions can be issued using the protocol, the user must register the smartphone with the bank. In addition, the bank supplies the

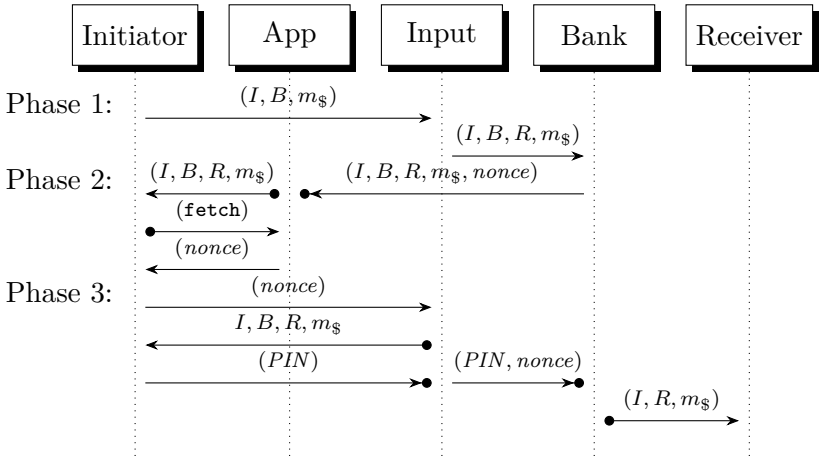


Figure 3.1: Protocol flow of L-Pay using authenticated and confidential channels drawn as $\bullet\!\!\longrightarrow$ and $\longrightarrow\!\!\bullet$ respectively.

user with a PIN. Both of these tasks can be carried out in a registration ceremony at a branch bank.

The protocol flow of L-Pay consists of three phases as shown in Figure 3.1. In the first phase, the initiator enters the desired transaction details into an input device such as an ATM or point of sale (POS) device. In this scenario, we assume that a user withdraws money at an ATM. The transaction details consist of the account number I of the initiator, the bank B, and the desired amount $m_\$$ to withdraw. The ATM forwards this information to the bank and adds the identifier of its money dispenser R. In the second phase, the transaction has to be confirmed using the smartphone. To achieve this, the bank transmits the transaction data and a nonce to the smartphone app. However, since our protocol should work with a smartphone that does not have an active internet connection, the data is relayed through the ATM and transmitted from the ATM to the smartphone using a QR code. Because the transmitted information is confidential, the bank encrypts the data using the public key registered for the app during registration. The app decrypts the message and displays the transaction data. After the user confirmed the transaction data, the app displays the transmitted nonce. It is important that the app does not display the nonce before the user confirmed the transaction data. Otherwise, the displayed nonce could be picked up by a CCTV camera monitoring the ATM and used by an adversary to confirm the transaction without the user's explicit consent. In the last phase, the user confirms the transaction at the ATM by entering the nonce and the PIN. The ATM forwards the PIN and nonce to the bank. Finally, the bank verifies that the nonce belongs to a pending transaction of an account and that the PIN is valid for the account. In this case, the bank notifies the money dispensing unit inside the ATM to dispense the requested amount of cash.

3.4.1 Communication Channels

L-Pay relies on multiple communication channels between the participants of the protocol. In the following, we list the necessary channels and describe how they are realized in L-Pay.

(i) Bank $\longrightarrow\!\!\bullet$ App: confidential channel

(ii) Initiator $\bullet\!\!\longleftrightarrow\!\!\bullet$ Smartphone: bidirectional authenticated channel

(iii) Initiator $\longrightarrow\!\!\bullet$ ATM: confidential channel

(iv) ATM $\longrightarrow\!\!\bullet$ Bank: confidential channel

(v) Bank $\bullet\!\!\longrightarrow$ Receiver: authenticated channel

The channel between the bank and the smartphone app must be confidential to protect the transmitted nonce. This can be realized using public-key encryption. In addition, we need an authenticated channel between the initiator and the smartphone to assure that only the initiator can confirm the transaction. This is commonly realized by the lockscreen mechanism of the smartphone. Furthermore, the initiator needs to enter the PIN into the ATM confidentially, e.g., by covering the PIN pad with one's hand, such that it cannot be eavesdropped by a CCTV camera. In addition, the ATM needs a confidential channel to send the PIN to the bank. Otherwise, an attacker controlling the network between the ATM and the bank can learn the PIN. Furthermore, the transmission of the PIN has to be resilient to replay attacks. Encrypting the PIN and sending it to the bank is not sufficient, because an attacker controlling the network can replay the recorded ciphertext in a subsequent transaction. In L-Pay, the PIN and the nonce are encrypted together in the same message and sent to the bank. Because the nonce is only valid for one specific transaction, the ciphertext cannot be reused to confirm different transactions. Finally, we need an authenticated channel between the bank and the receiver to inform the receiver of the completed transaction. For this channel, it is important that messages are only valid for a short amount of time. An attacker controlling the network can delay messages to the receiver. This could be abused to delay a command to dispense cash. An honest user will probably wait some time for the money dispenser to cash out money but will eventually leave and contact the bank. In the meantime, the attacker can deliver the command and collect the user's money. To prevent this, the message to the money dispenser should include a timestamp such that the receiver can verify that the message is fresh and drop expired messages. However, this also means that the receiver must have access to a hardware clock. The use of sequence numbers is not sufficient on its own as this only prevents reordering messages but not delaying a message.

3.5 A Formal Model for L-Pay

To reason about the security of our protocol, we formalize L-Pay, as well as our security notion *one-out-of-two security*. In previous work, we presented a formal model for electronic payment based on the Universal Composability (UC) framework [1]. However, in this thesis we present a formalization of the L-Pay protocol and security proof using Tamarin Prover (Tamarin) [125]. Tamarin is a tool for the computer-assisted formal verification of cryptographic protocols. For a brief introduction to Tamarin, we refer to Section 2.5.3. Using Tamarin to formally verify the security properties of L-Pay provides the following advantages:

(i) The proof generation is automated. Once the protocol and the security notion are formalized, Tamarin can be used to generate a proof. This eliminates the potential of human errors during proof generation. Furthermore, Tamarin outputs a counterexample if the prover reasons that the protocol does not fulfill the security notion. This constitutes a possible attack on the protocol.

(ii) We model a stronger attack model incorporating adaptive corruption. That means, the attacker can compromise a device at any time during the protocol execution. The UC model in [1] is based on the concept of static corruption, where the adversary can only compromise devices prior to the protocol execution.

(iii) Manual proof generation quickly gets unmanageable for complex scenarios. Thus, using Tamarin we were able to model a more complex scenario incorporating multiple banks and automated teller machines (ATMs).

(iv) By providing two security proofs, we strengthen the confidence in our protocol.

3.5.1 Assumptions

We do make certain assumptions regarding the trustworthiness of different protocol participants. As discussed before, we assume the money dispensing unit inside an ATM to be trusted. Since our work focuses on the challenges that arise from the interaction of humans with untrustworthy devices over insecure communication, we do not model the banks' book-keeping and assume the banks to be incorruptible.

Our model for electronic payment is thus designed with regards to the following principles:

(i) The adversary always gains access to all transaction data. An electronic payment operation can be secure (that is all participants of

the transaction get notified about the correct and non-manipulated transaction data) without the transaction data being secret.

(ii) The payment operation occurs in three stages. In the first stage, the initiator inputs his intended transaction data which the adversary can change at will. This models that a corrupted input device will always be able to change the human initiator's transaction data, even if it will be detected at a later stage. In the second and third stage, the bank and the receiver are notified about the transaction data.

3.5.2 Modeling L-Pay using Tamarin

We use Tamarin to model our protocol L-Pay. Our model supports adaptive corruption, i.e., the adversary can compromise ATMs and smartphones at any time during protocol execution. Furthermore, our model supports scenarios with multiple banks and ATMs. We do not consider the compromise of a bank and thus assume the banks to be incorruptible and to be able to communicate securely with each other. We assume that an ATM can be used by customers of all registered banks and that the ATM can communicate confidentially with these banks. Furthermore, a user can open bank accounts at multiple banks. We use the following multi-set rewriting rules to model L-Pay using Tamarin:

- create_bank: Registers a new bank as a persistent fact.
- create_atm: Registers a new ATM as a persistent fact.
- init_app: A user installs the banking app required for L-Pay on the smartphone. The app generates a key pair that is later used during registration.
- register_account: Register a new bank account for a user at a specific bank. The user provides the bank with the public key of his banking app. Furthermore, the bank assigns a random Personal Identification Number (PIN) to the user.
- init_transaction: A honest user commences a transaction at an ATM. After entering the transaction details, the ATM forwards the transaction data to the bank.
- attacker_init_transaction: A malicious user commences a transaction at an ATM and enters arbitrary transaction details. This is modeled as a separate rule, because it allows to differentiate transactions initiated by honest and malicious users, which is important for the security notion.

- bank_receive_transaction: A bank receives a transaction from an ATM. The bank verifies that the bank account exists and transmits the transaction details in conjunction with a nonce to the app registered for the account.

- app_receive_transaction: The app receives and decrypts an encrypted message containing the transaction details and the nonce from the bank.

- user_verify_app_transaction: The user verifies the transaction details displayed by the app. If the user confirms the transaction, the app displays the nonce transmitted by the bank.

- atm_user_enter_pin_and_nonce: The user verifies the transaction details displayed by the ATM and confirms the transaction by entering his secret PIN and the nonce displayed by the app.

- atm_attacker_enter_pin_and_nonce: A malicious user tries to confirm a transaction at an ATM by entering a previously acquired PIN and nonce.

- bank_verify_transaction: The bank receives the nonce and the PIN from the ATM via a confidential channel. The bank verifies the nonce, as well as the PIN of the user. If everything is correct, the transaction is confirmed by the bank.

- compromise_atm: An ATM is compromised by an adversary.

- compromised_atm_leak_pin: A user confirms a transaction at a compromised ATM. The adversary thus learns the entered PIN.

- compromise_app: The smartphone of a user is compromised by an adversary. The adversary leaks the private key stored in the app.

Figure 3.2 shows a simplified trace of rules for creating a bank account and executing a transaction. The trace is separated into the different phases of the L-Pay protocol. The figure also includes rules that allow an attacker to interfere with the protocol such as compromising a device.

3.5.3 How Our Model Captures Existing Attacks

One of our main motivations for establishing a formal model for electronic payment is to make trust assumptions explicit in order to detect unrealistic ones which enable practical attacks like [20], [129] and [57]. Thus, our model needs to be able to capture these kinds of attacks. In the following, we explain how this is achieved.

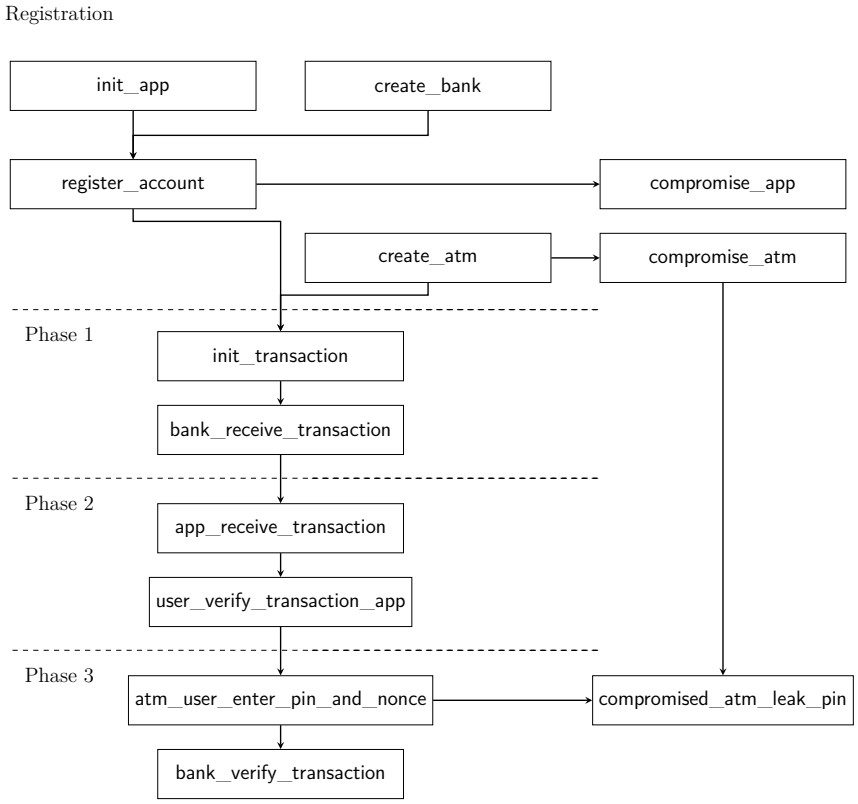


Figure 3.2: Simplified execution trace of our Tamarin model for initiating and confirming a transaction at an ATM with L-Pay.

Changing Transaction Data

An adversary controlling the input device can easily change transaction data. This is represented in our model by transmitting the transaction data over an insecure channel to the bank, which can be manipulated arbitrarily by the adversary.

Relay Attacks

The aim of a relay attack [57] is to get Alice to authorize an unintended transaction, which benefits the attacker, by relaying legitimate protocol messages between the point of sale (POS) device she uses to pay for goods to another POS device. Our model supports relay attacks out of the box. Since Tamarin is based on the Dolev-Yao attack model, all messages passed through the network can be read, manipulated, forwarded and replayed by the adversary. Furthermore, in our model messages from an ATM to the bank are not authenticated and can be forwarded arbitrarily.

```
lemma only_user_initiated_transactions_accepted:

"All acc bank data atm #i.
BankVerifiedTransaction(acc, bank, atm, data) @ #i
==>
(
Ex #j. (TransactionInitiated(acc, bank, atm, data) @ #j)
)"
```

Listing 3.1: First security lemma that states that all accepted transactions are initiated by an honest user.

Pre-Play Attacks

Pre-play attacks [20] basically rely on two facts: (i) once unlocked, smartcards, as used in the EMV protocol, can be coerced into generating message authentication codes (MACs) for arbitrary transaction messages and (ii) that even honest ATMs use predictable "unpredictable numbers". Cards interacting with a corrupted ATM can be used to easily generate additional MAC tags. Since in our protocol, the smartcard is replaced by a smartphone and instead of generating a MAC, the smartphone decrypts a nonce transferred from the bank, the protocol enforces freshness and resists injecting pre-calculated data. Nonetheless, this kind of attack could be represented in our model by adding a rule that grants an adversary access to a signature oracle once a smartcard is inserted into the input device.

3.5.4 Modeling One-out-of-two Security

In addition to the protocol, we model our security notion *one-out-of-two security* using first-order logic formulas for Tamarin to verify. We need to model two aspects of our security notion:

(i) The security goal

(ii) The attack model

The definition of the security goal is inspired by [115] and is split into two lemmas. The first lemma *only_user_initiated_transactions_accepted* shown in Listing 3.1 states that each transaction that is accepted by a bank must have been initiated by an honest user. This lemma ensures that an adversary is not able to execute arbitrary transactions.

However, this lemma does not prevent replay attacks. Consider a simple protocol where the initiator sends a signature of the transaction data consisting of the bank, identifier of the ATM, and amount to the bank. Once the transaction is complete, the attacker can replay the recorded signature to execute the same transaction again. This does not falsify the lemma but

```
lemma replay_attack_impossible:

"All acc1 bank1 data1 atm1 acc2 bank2 data2 atm2 #i #j.
BankVerifiedTransaction(acc1, bank1, atm1, data1) @i &
BankVerifiedTransaction(acc2, bank2, atm2, data2) @j &
not #i = #j
==>
( Ex #k #l.
TransactionInitiated(acc1, bank1, atm1, data1) @k &
TransactionInitiated(acc2, bank2, atm2, data2) @l &
not #k = #l)"
```

Listing 3.2: Second security lemma that states that transactions cannot be replayed.

is clearly undesirable. Thus, we define an additional lemma *replay_attack_impossible* as shown in Listing 3.2 that captures exactly this requirement. It states that for each possible pair of accepted transactions there must exist two unique points in time when each of them was initiated by an honest user.

Besides the security goal, our security notion must respect our attack model too. For *one-out-of-two security*, we allow the adversary to either compromise an ATM or the user's smartphone but not both. However, our formalization of L-Pay allows the adversary to compromise ATMs and smartphones participating in the protocol arbitrarily. Thus, we have to restrict compromise of devices in our security notion explicitly. In our model, compromise of a device results in an action fact being written to the trace. For example, if a smartphone is compromised, the action fact *CompromiseApp* is written to the trace. For ATMs, our model establishes a persistent fact *ATM_Compromised*. Once this fact is established, an attacker might leak PINs entered by users that withdraw money at the compromised ATM. If the PIN of a user is leaked by an ATM, we output the action fact *WithdrawAtCompromisedATM*.

To incorporate our attack model, we add the following clause to the security lemmas. Thereby, we state that the lemma must hold as long as the adversary does not compromise the user's app and an ATM where the user withdraws money.

```
Ex #j #k. CompromiseApp(acc, bank) @#j  &
    WithdrawAtCompromisedATM(acc, bank) @#k
```

We show the complete definition of our security notion in Listing 3.3. The entire model of our protocol and the security notion can be found in Appendix A. A protocol satisfies *one-out-of-two security* if both lemmas *only_user_initiated_transactions_accepted* and *replay_attack_impossible* are fulfilled. We used the Tamarin prover to verify that our protocol L-Pay

```
lemma only_user_initiated_transactions_accepted:

"All acc bank data atm #i.
BankVerifiedTransaction(acc, bank, atm, data)@ #i
==>
(( Ex #j.
TransactionInitiated(acc, bank, atm, data) @ #j)
| (Ex #j #k. CompromiseApp(acc, bank) @#j &
           CompromiseATM(atm) @#k))"

lemma replay_attack_impossible:

"All acc1 bank1 data1 atm1 acc2 bank2 data2 atm2 #i #j.
BankVerifiedTransaction(acc1, bank1, atm1, data1) @i &
BankVerifiedTransaction(acc2, bank2, atm2, data2) @j &
not #i = #j
==>
(( Ex #k #l.
TransactionInitiated(acc1, bank1, atm1, data1) @k &
TransactionInitiated(acc2, bank2, atm2, data2) @l &
not #k = #l
)
| (Ex #m #n. CompromiseApp(acc1, bank1) @m &
   WithdrawAtCompromisedATM(acc1, bank1) @n)
| (Ex #m #n. CompromiseApp(acc2, bank2) @m &
   WithdrawAtCompromisedATM(acc2, bank2) @n))"
```

Listing 3.3: Formalization of our security notion one-out-of-two security for electronic payment using two Tamarin lemmas.

fulfills both lemmas. Thus, L-Pay achieves *one-out-of-two security* and is secure as long as one of the two devices is not compromised.

3.6 Related Work

In this section, we introduce different types of related work, including secure human-server communication, alternative hardware assumptions, electronic cash, and the EMV standard.

3.6.1 Secure Human-Server Communication

Basin, Radomirovic, and Schläpfer [14] give an enumeration of minimal topologies of channels between a human (restricted in its abilities), a trusted server, a possibly corrupted intermediary and a trusted device, that realize an authenticated channel between the human and the server. Our work differs in two main aspects: Their model uses either fully secure or untrusted

channels only and cannot account for just authenticated or just confidential communication, which is important in our setting. For example, we assume that information displayed on the user's smartphone is not confidential due to the presence of CCTV cameras and shoulder-surfing. Second, their solution is based on a trusted device, whereas in our work no single device needs to be trusted (see Section 3.2.3).

3.6.2 Alternative Hardware Assumptions

As we identified in Section 3.2.2, the *confirmation of payment information* by the user is an important sub-problem we aim to solve for achieving secure payment. A possible solution is "Display TAN" [76] providing a smartcard with a display to show the transaction data. Smart-Guard [51] uses such smartcards with a display together with an encrypting keyboard fixed to the card to achieve a functionality which may be used for payment. These strong hardware assumptions allow for flexible trust assumptions, accounting for several combinations of trusted/compromised status of the involved devices. For our construction we do not propose a new kind of hardware device but rely on the user's smartphone.

3.6.3 Electronic Cash and Cryptocurrencies

Electronic cash was invented to support the privacy properties of cash money in electronic payment systems [35, 34]. These systems offer untraceable payments that hide the payment details from third parties. At the same time, they ensure that double spending is not possible, i.e., the bank can identify the corresponding user if a digital coin is spent multiple times. Modern decentralized cryptocurrencies such as Bitcoin establish an electronic payment system that prevents double-spending without relying on a trusted bank [75]. This is achieved by storing transactions in a public ledger and using a consensus mechanism based on proof-of-work. In general, electronic cash systems and cryptocurrencies have very different design goals. They aim for untraceable payments or the replacement of a trusted bank by a distributed system. In contrast, we are concerned with the authenticated transmission of the transaction data from a human user to the bank.

3.6.4 EMV

EMV is not only a single payment protocol, but a complete protocol suite for electronic payment (cf. [61, 62, 63]). Protocols that are EMV-compliant might just implement the EMV interface while using another secure protocol. This means that, while there are multiple attacks against the EMV *payment protocol*, not every protocol with EMV in its name is automatically insecure. In addition to the attacks mentioned previously, there are other attacks as described by Chothia et al. [37] and Emms et al. [60].

Degabriele et al. [50] investigate the joint security of encryption and signatures in EMV using the same key-pair. A scheme based on elliptic curves (as it is used in EMV) is proven secure in their model. However, as they conclude, their proof does not eliminate certain kinds of protocol-level attacks. Cortier et al. [43] present an EMV-compliant protocol for point of sale (POS) using a secure element in a mobile device and prove the security of their protocol using Tamarin [125]. In contrast, our protocol does not rely on a single trusted hardware component but instead spreads trust between two involved devices. Basin, Sasse, and Toro-Pozo [13] model several configurations of the EMV standard using Tamarin and discover new flaws. However, their model is based on the Dolev-Yao attack model and does thus not consider compromise of the POS device or automated teller machine (ATM).

3.7 Conclusion and Future Work

Designing payment protocols that protect against infected devices poses a particular challenge. They typically involve a human user who is not capable of performing cryptographic operations and therefore needs an intermediate device (e.g. an automated teller machine (ATM)) to interface with the protocol, which might be compromised. For payment protocols involving additional devices such as smartphones we proposed *one-out-of-two security* as a remedy. Protocols fulfilling this security notion are secure as long as either the ATM or additional device is uncompromised. Based on these results, we examined current payment protocols and found that most do not realize this notion. Online banking protocols such as chip authentication program (CAP) and photoTAN might satisfy the requirements for secure electronic payment. However, online banking differs from electronic payment in many aspects and is thus analyzed separately in Chapter 4. We designed a protocol called L-Pay (inspired by online banking protocols), which uses an additional smartphone and is secure even if either the ATM or the smartphone is compromised. To verify the security properties of our protocol, we formalized L-Pay and verified that it fulfills *one-out-of-two security* using Tamarin Prover (Tamarin).

Chapter 4

Web Authentication

The contents of this chapter are based on joint work with Benedikt Wagner, Julian Herr, Jochen Rill, Marek Wehmer, Niklas Goerke, and Ingmar Baumgart. Parts of the content previously appeared in other publications. The publications are listed below.

- Timon Hackenjos, Benedikt Wagner, Julian Herr, Jochen Rill, Marek Wehmer, Niklas Goerke, and Ingmar Baumgart. "FIDO2 With Two Displays—Or How to Protect Security-Critical Web Transactions Against Malware Attacks". In: *arXiv preprint arXiv:2206.13358* (2022). DOI: 10.48550/arXiv.2206.13358 [91]

4.1 Introduction

In recent years, the World Wide Web and the multitude of different services offered within changed almost everyone's life. Whether we want to send an email, do online banking, buy a product, or update our social media profile, we use the Web. In general, each of these activities requires a separate account with which we authenticate ourselves. According to a study by Google in partnership with The Harris Poll in 2019, the average American has 27 different online accounts [7].

Passwords always were and still are the main means of authentication on the Web. However, since one cannot possibly remember that many different passwords (and the adoption of password managers is still very low) re-using passwords on a multitude of different sites is a common practice [49]. This means that one stolen password from an account on an unimportant website might also give an adversary access to more important ones, like an online banking account.

As more and more of our private and professional life started to happen on the Internet, compromising accounts through simple and cheap attacks like credential stuffing and phishing became increasingly more attractive and lucrative for attackers.

To mitigate these risks, security experts started to advertise the use of two-factor authentication instead of only a single password [105]. A mandatory second authenticating factor (like a one-time password (OTP)) severely restricts the utility of stolen passwords for an attacker. Users and account providers alike took a very long time to adopt this recommendation, but nowadays many web applications offer at least one form of two-factor authentication. Most implementations require the user to present a second factor together with a password during login. If successful, the user then receives an authenticated session token (stored within a cookie), which she can use to interact with the site and her account without having to authenticate herself again for a while.

When using two-factor authentication, it is recommended to use two of the three following factors: something you know, something you have and something you are. Behind this categorization lies the assumption that it is improbable for the same adversary to compromise factors from different categories (e.g., an adversary that gets a password from a password leak cannot also steal a copy of the user's fingerprint). However, this is only true in a very specific adversarial model. An attacker who has compromised a victim's computer does not need to steal a copy of the fingerprint; he can simply manipulate all interactions with a website by using the authenticated session token stored in the browser after the victim performed a legitimate login.

Indeed, two-factor authentication as it is used today does not help against a number of attack techniques used by real-life adversaries [146, 5, 39]. Most schemes are susceptible to malware attacks and some popular forms, like OTP, are also vulnerable to real-time phishing, in which an adversary relays authentication details from a fake website to a legitimate one [106, 115].

Bruce Schneier adequately summarized the applicability of two-factor authentication in as early as 2005: "Two-factor authentication isn't our savior. It won't defend against phishing. It's not going to prevent identity theft. It's not going to secure online accounts from fraudulent transactions. It solves the security problems we had 10 years ago, not the security problems we have today" [150].

In recent years, online banking has been moving into the right direction security-wise by introducing *transaction authentication*, which is the verification of transaction details (often on an additional device, but not necessarily). Authenticating transactions individually mitigates the risk of session hijacking by stealing a cookie. Furthermore, manipulation of transaction details can be detected if one is using an additional device to check them.

The chip authentication program (CAP) protocol used in online banking thus provides a high level of security [95]. It relies on a dedicated card reader with a display. The device offers a very small attack surface for infection by malware as it has limited functionality and is specifically built for this use case.

Carrying an additional device has been identified to be unpleasant for many users [116, 120]. In consequence, many banks abandoned dedicated hardware tokens and transitioned to app-based authentication schemes such as photoTAN. However, similar to regular computers, smartphones have large attack surfaces, are susceptible to, and have been attacked by malware [68]. By compromising the smartphone, attackers can bypass confirmation of transactions on the device [95]. Therefore, if an attacker gets access to the user's password, he can access the online banking system and execute arbitrary transactions (see Section 4.2). Some banks even allow initiating transactions from the same device [95]. Thus, most current online banking schemes, which are required by law to offer strong security [66], do not adequately protect against malware attacks, since the smartphone needs to be fully trusted. Besides online banking, many other use cases such as administration panels and electronic health records handle security-critical transactions (see Section 4.6). However, most of them do not implement *transaction authentication* and thus do not protect against malware attacks.

The recent FIDO2 standard provides a widely implemented browser API called WebAuthn that simplifies integration of secure web authentication mechanisms relying on public-key cryptography and supporting authentication of individual transactions. However, authentication with FIDO2 as it is used today does not involve a second device with a display and thus suffers from susceptibility to malware attacks as described before. An extension for *transaction authentication* has even been removed from the latest version of the WebAuthn standard because it was not implemented in any browser[1].

We show how to design web authentication schemes using two devices (one of which could be a smartphone) which protect security-critical transactions, even if one device is fully compromised. While this might seem like an impossible task, it can be done. In the following, we show how.

4.2 Attacks on Web Authentication

In this section, we analyze attacks on current web authentication schemes and introduce our attack model that serves as a basis for the following considerations.

4.2.1 Password Attacks

Password authentication is by far the most prominent authentication scheme in the web. The main problems of password authentication are that users choose weak passwords and reuse them across multiple sites [155]. Brute-force and password spraying attacks exploit weak passwords while credential-stuffing attacks focus on reused passwords. For example, the video conferenc-

[1] `https://github.com/w3c/webauthn/issues/1386`

ing solution Zoom was hit by a credential-stuffing attack and a large number of accounts were compromised [103]. The success of these attacks is not surprising considering the amount of publicly available credentials [102].

Risk-based authentication (RBA) aims to strengthen password authentication by monitoring features such as the IP address and the user agent of the browser and triggering additional authentication during login if they differ from those recorded before [162]. While this limits the impact of a stolen password, most features are easy to detect and spoof during a phishing or malware attack. Criminal platforms evolved that sell access to user profiles containing credentials and features that allow bypassing RBA [26].

4.2.2 Real-Time Phishing

In 2018, Amnesty International reported targeted phishing attacks on Google and Yahoo accounts that bypassed one-time password (OTP)-based two-factor authentication [5]. By automating the process of using the stolen password and OTP, the attackers were able to work around the short validity of the token. Despite being more complex than classic phishing attacks that only harvest passwords for later use, real-time phishing attacks are not sophisticated. Several tools are publicly available to automate this attack [121, 133]. In addition, real-time phishing attacks can be used to identify features of the user necessary to bypass RBA [26].

4.2.3 Malware

As described in prior work, multiple ways exist to remotely infect devices with malware ranging from drive-by downloads and email attachments to social engineering attacks that convince a user to install malware herself [83, 131]. Google's Threat Analysis Group even detected the use of a zero-day exploit for the Safari browser to steal cookies for popular websites such as Google, Microsoft and LinkedIn [153]. Modern operating systems support to isolate applications from another which might limit the impact a piece of malware can have on a system. However, these mechanisms have been bypassed multiple times in the past, e.g., using privilege escalation exploits. Thus, we assume malware to run with the highest privileges available on a system. For more information regarding isolation mechanisms of modern operating systems we refer to Section 6.2.1. Of course, smartphones are affected by malware too [68]. In the following, we discuss two types of attacks that can be carried out by malware to issue a malicious transaction.

Transaction Manipulation

In the following, we assume that an authentication scheme is used that does not display transaction details on an additional device. Malware on the primary device can then carry out a transaction manipulation attack as

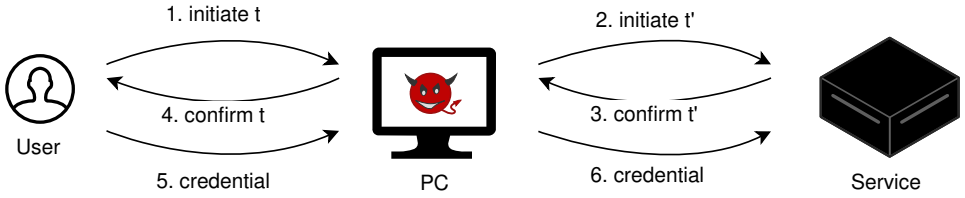


Figure 4.1: Transaction manipulation attack: malware on the device manipulates the transaction data t and issues a transaction t' instead.

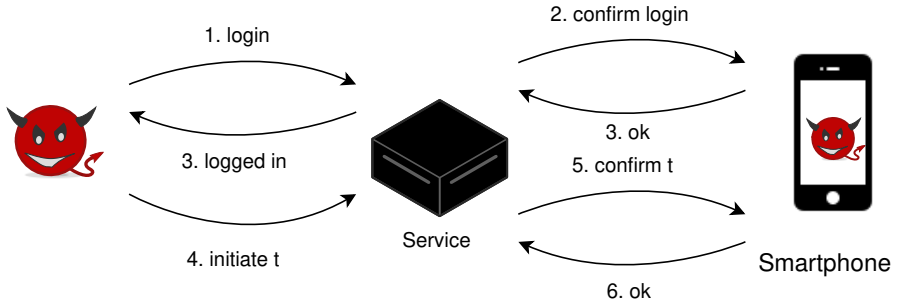


Figure 4.2: Transaction initiation attack: malware on a smartphone confirms transaction initiated by the attacker.

depicted in Figure 4.1. Suppose that a user wants to make a security critical transaction on a website. After logging in (potentially with two factors), she enters the desired transaction details t (1), however, the compromised device initiates a manipulated transaction t' (2) instead. If the service relies on session authentication only, the manipulated transaction t' is authenticated by the cookie sent by the browser and thus confirmed immediately. However, the service might require confirmation by a second factor for transaction t' such as an OTP (3). In this case, the malware prompts the user to confirm the original transaction t (4). As the user cannot detect the manipulation, she supplies the required OTP (5). The attacker can use the OTP to confirm the manipulated transaction t' (6). This attack applies to other authentication schemes such as FIDO2 as well. Transaction manipulation has been used extensively in the wild by online banking malware such as ZeuS and SpyEye [146]. Note that RBA does not prevent this attack, as the transaction is initiated in a valid user session.

Transaction Initiation

Authentication schemes that incorporate an additional device for transaction verification are often susceptible to a transaction initiation attack as depicted in Figure 4.2. We assume that the attacker compromised the additional device and installed malware. In a next step, the attacker needs to determine

the user's password using one of the methods from Sections 4.2.1 and 4.2.2. Potentially, the password is even stored on the additional device and thus accessible to the attacker. With the password, the attacker logs in to the service from his own device using the victim's account (1). The service might require confirmation of the login attempt on the additional device, e.g., because the service implements RBA and detected a change in the user profile. Usually, confirmation of the login attempt is implemented using text messages, email or an app [163]. However, since the attacker remotely controls the additional device, he can confirm the login request (3) [95]. After logging in successfully, the attacker can initiate an arbitrary transaction t (4). The service might require confirmation of the transaction (5), however, again the attacker controlling the additional device can confirm the request (6).

4.2.4 Our Attack Model

We identify the following three main means of attack on web authentication schemes:

- Password attacks
- Real-time Phishing
- Malware

Malware attacks are more powerful than real-time phishing and password attacks. Password attacks give access to passwords only. In addition, real-time phishing grants access to other credentials entered by the user such as an OTP. The strongest type of attack is compromising a device and infecting it with malware. This grants full access to input and output interfaces of a device such as the keyboard and display, as well as credentials stored on the device. Thus, malware on a device might eavesdrop on credentials entered by the user, as well as manipulate information displayed on the screen.

We assume malware to be executed with the highest user privileges available such as root or administrator and thus malware might manipulate the operating system. This assumption is supported by prior work that documents the use of privilege escalation attacks in the wild [68]. Furthermore, a similar model for malware has been used in prior work [115, 95].

Because of the high prevalence of password and phishing attacks [155], we assume all attackers to be able to carry them out. However, we differentiate attackers by their ability to infect involved devices with malware. The underlying premise is that each device that has to be compromised by the attacker to break a scheme increases the difficulty of the attack. Furthermore, we consider that an attacker can carry out multiple attacks targeting the same user. For example, an attacker that infects a smartphone with malware can also mount a password or phishing attack targeting the account of the user.

4.3 Malware-Resistant Web Authentication

As discussed in the last section, a secure web authentication scheme should protect against real-time phishing and malware attacks. In the following, we argue that the concept of *transaction authentication* is inevitable to protect against malware attacks and that *two-factor authentication*, as it is currently defined, does not provide security benefits in this attack model.

4.3.1 Transaction Authentication

Usually, users *sign in* to a web service before using it. During this process, the web service authenticates the user and establishes a trusted session. *Session authentication* means that once the session is created, the user is allowed to perform an arbitrary number of actions by sending requests to the service. It can be described as authenticating the session and then assuming every action performed as part of the session has been authorized by the user.

On the other hand, using *transaction authentication*, transactions are initiated in an authenticated session, but confirmation of each individual transaction is required (e.g., verification of transaction details by the user on a separate device). Nonetheless, *transaction authentication* is usually used for security-critical transactions only. For example, in online banking, transferring money between a checking and a savings account of the same user could rely on *session authentication*. Security-critical transactions such as bank transfers to an external account should require *transaction authentication*. These transactions must be identified for each use case on its own. We provide examples for such transactions in Section 4.6.

Relying on *session authentication* only is fundamentally flawed in the presence of malware. Taking control of the session lets an attacker interact with the service on behalf of the user, independently of the used authentication procedure. Thus, strong security guarantees can only be stated for transactions protected by *transaction authentication*. However, *transaction authentication* is not sufficient to protect against malware attacks on its own. For example, if transactions are initiated and confirmed on the same device, malware can manipulate the transaction data without the user noticing [95] Combined with other measures proposed below, individual transactions can be protected to make it impossible for an attacker to perform an unsolicited transaction without the user explicitly authenticating it.

4.3.2 Flaws in Two-Factor Authentication

The notion of two-factor authentication is too weak in a realistic attack model and does not imply any security guarantees in the presence of a malware or phishing attacker, even for individually authenticated transactions.

Intuitively, we expect a two-factor authentication scheme to be secure as long as one of the two factors is not compromised. However, this is satisfied neither by the notion nor by currently deployed schemes in the presence of malware. As described in Section 4.2, malware and real-time phishing attacks are not prevented thoroughly by current two-factor authentication schemes.

Two-factor authentication schemes used in online banking such as the chip authentication program (CAP) protocol and photoTAN provide resistance to malware on the device running the browser. However, the reason for this does not lie in the fact that they fulfill the two-factor authentication definition, but that they require the user to verify transaction details on a separate device. Thus, we need a new way of defining two-factor authentication that fulfills the intuitive promise of the notion: ensuring the security of a scheme even if one of two factors is compromised. This redefinition of two-factor authentication makes malware and real-time phishing attacks infeasible and thus solves the weaknesses of current web authentication schemes.

4.3.3 One-out-of-two Security for the Web

In Chapter 3 on secure electronic payment, we formalized exactly this requirement. However, instead of relying on two factors from the three categories, we separate trust between two independent devices. Our electronic payment protocol L-Pay uses two devices and is still secure (with regards to specific security properties) even if one of the devices is fully compromised. We called this security notion *one-out-of-two security* and showed that it can be fulfilled by requiring the user to verify the transaction details on both devices. In this chapter, we adapt *one-out-of-two security* to web authentication by letting the user verify details of individual transactions on two separate devices.

L-Pay relies on a so-called *confirmation mechanism* to fulfill one-out-of-two security. The transaction details are displayed to the user (e.g., on an additional device) and explicit confirmation is required before performing the transaction. We reuse the idea of a confirmation channel in the design of our web authentication scheme. However, we use a slightly stronger version of one-out-of-two security. Our security notion also covers attackers that mount multiple attacks to target a user, e.g., infecting a device with malware and executing a brute-force attack on the password. According to our definition, a protocol that uses two separate devices exhibits one-out-of-two security, if an attacker who has the capabilities defined in our attack model in Section 4.2 is not able to issue a fraudulent transaction without compromising both devices. To make it harder for an attacker to compromise both devices, the devices should be hardened and isolated from each other. Attacks have been described that abuse synchronization mechanisms to compromise a smartphone from an infected computer [114]. This illustrates the importance

of proper device isolation. However, details about how device isolation can be achieved goes beyond the scope of this work.

Note that the goal of the security notion is authenticity of transactions and not confidentiality of the transaction details. To be able to confirm the authenticity of a transaction on two devices, the user must either see or enter the details on each device separately. In either case, both devices will need to handle the transaction details in plain text. Thus, compromising one device is enough to violate the confidentiality of transaction details. As described in Section 4.6, by limiting access to actions that return or change data on the service, the authenticity guarantee of transactions can be used to achieve *integrity* and *confidentiality* of data stored on the server.

4.3.4 Blueprint for Secure Web Authentication

Constructing a web authentication scheme that fulfills one-out-of-two security and is thus only vulnerable to attackers that are able to compromise both devices is not an easy task. As we will show in Section 4.7, commonly used schemes do not have this property.

We propose Two Display Authentication (2DA), a blueprint for secure web authentication. Following the blueprint simplifies constructing schemes that fulfill one-out-of-two security. We identify the following requirements for 2DA.

(2DA.1) Authenticate security-critical transactions.

(2DA.2) Require verification of transaction details by the user on both devices.

(2DA.3) Use authentication mechanisms that protect against network-based attacks.

(2DA.4) Use local authentication mechanisms to limit access to the devices.

The first requirement addresses the fact that the security guarantees can only be accomplished for transactions that are authenticated individually. The second requirement is necessary to thwart attacks that involve one of the two devices being compromised. Entering the transaction details on a device instead of reading them from the display is also possible. Furthermore, the third requirement relates to the communication between the devices and the server. If an insecure authentication protocol is used that can be hijacked on the network level, then an attacker would not bother to compromise a device to impersonate the user. For example, using only a password to authenticate a transaction, guessing the password or mounting a phishing attack is sufficient to impersonate the user without the necessity of compromising the primary device. Thus, one-out-of-two security cannot be

achieved by using password authentication. Finally, the fourth requirement protects devices in case they are stolen. Even though this is not strictly necessary to fulfill one-out-of-two security, as the notion does not consider how a device is compromised, we deem it an important mechanism to prevent physical attackers from compromising or abusing stolen devices. Other hardening measures that increase the difficulty of compromising a device can be used to further enhance security, e.g., enabling exploit mitigations.

Note that a secure transaction authentication scheme does not protect a user from authenticating an unintended transaction if the original transaction data was manipulated in the first place. Thus, it is of utmost importance that the transaction data is either self-explanatory or the user has the ability to compare the transaction data with a known safe value. For example, attacks on online banking have been described that are based on manipulating digital invoices [93]. If the original transaction details are only available on one device, then no scheme can offer protection when the device is compromised.

The goal of 2DA is to simplify creating web authentication schemes that protect security-critical transactions by relying on *one-out-of-two security*. In the following section, we show that building on existing web authentication mechanisms such as FIDO2, we can use this blueprint to design an authentication scheme that is secure, even in the presence of malware and real-time phishing attacks.

4.4 Two Display Authentication using FIDO2

We use our blueprint 2DA from the last section to design FIDO2 With Two Displays (FIDO2D), a new web authentication scheme that fulfills one-out-of-two security, i.e., the scheme is secure as long as one of the two devices is not compromised. We refer to Section 4.5 for a detailed security proof. On a high level, a user U aims to initiate a transaction at a server S using a computer B running a browser. The user also holds an additional device A, such as a smartphone.

We assume that (2DA.4) is satisfied by existing mechanisms that restrict access to B and A. Most devices offer to setup a lockscreen that requires the user to authenticate herself to the device before getting access, e.g., using a Personal Identification Number (PIN). Thus, an attacker would have to steal a device and the corresponding authentication factor for the device. Compromising both devices by stealing them and their authentication factors is even more challenging. Optimally, the user has even setup individual authentication factors for the two devices. Following (2DA.3), we build on top of an existing authentication mechanism with appropriate security, and following (2DA.2) we use two instances of this mechanism. That is, one instance is executed between B and S, and one instance is executed between A and S. Again, following (2DA.2) we let the user verify the transaction details

during both instances. To be more precise, we organize both instances in a way that the user enters the transaction details in the browser on B and require verification of the transaction data by the user on A using a confirmation channel. In the following, we introduce the underlying authentication scheme FIDO2, and then describe the protocol for registration and authentication in its entirety.

4.4.1 FIDO2

We use FIDO2 as the underlying authentication mechanism [8]. FIDO2 is an interactive public-key challenge-response authentication protocol published by the Fast IDentity Online (FIDO) Alliance. It is used on top of Transport Layer Security (TLS). Both registration and authentication consist of a three-message flow between client and server. The first message from client to server initiates the interaction. The second message is sent from server to client and contains a randomly sampled challenge. Finally, the client sends a signature of this challenge and some additional data to the server. For more information about the basic protocol flow of FIDO2 we refer to Section 2.4.

More precisely, the client does not execute the cryptographic operations itself, but rather forwards messages to a so-called authenticator, which contains all key material. FIDO2 differentiates platform authenticators that are integrated into devices and roaming authenticators that can be connected via transports such as USB. For our further description, we focus on the FIDO client, because most authenticators do not contain a display to show transaction data themselves that is crucial for verification by U.

There are mainly two reasons for our choice of FIDO2. First, it is supported as an API by all modern browsers on the client side and libraries for various programming languages on the server side. This makes our protocol easy to implement and integrate. Second, it offers the security we need for (2DA.3) and (2DA.4). For our requirement (2DA.3), the authentication mechanism must resist network-based attacks such as phishing and replay attacks. FIDO2 is resilient to these attacks because both a random challenge and the identifier of the current web service, typically a domain name, is signed. Thus, the signature can neither be forwarded to another service because the identifier would not match nor replayed because a different challenge would be used. Furthermore, correctly guessing the used secret, i.e., the private key of a digital signature scheme, is very unlikely when an appropriate key length is used. For our requirement (2DA.4) FIDO2 supports a mechanism called *user verification*. When enabling *user verification*, the authenticator authenticates the user before allowing to use the stored key material, e.g., using a PIN. Thus, a stolen authenticator can only be used with the corresponding authentication factor. In the following, we describe our protocol FIDO2D and how it interfaces with FIDO2.

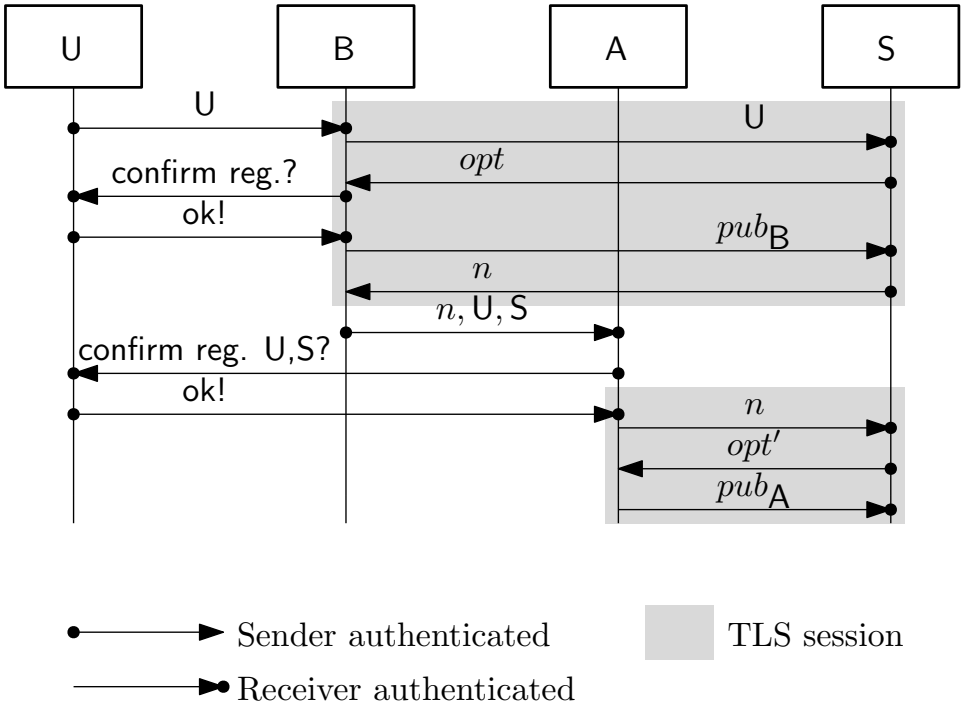


Figure 4.3: Sequence diagram for registration in our protocol FIDO2D. Here, a user U registers at a server S. The devices B and A are used to generate secret and public keys s_B, pub_B and s_A, pub_A during the process. The nonce n is randomly generated by S.

4.4.2 Registration

First, the user registers the device B using the standard FIDO2 registration ceremony. The device B and service S communicate with each other using a TLS session where the service authenticates itself to the clients using a certificate. The same applies to the communication between A and S later on. During registration, S stores a public key pub_B from B. Then, S provides a nonce to B to link the second device A to the account. The user transfers the nonce from device B to A using a QR code. Again, we use the FIDO2 registration ceremony to store a public key pub_A from A on S.

In more detail, the registration process is depicted in Figure 4.3. The server responds to a user registration request with a set of options *opt*, containing a random challenge, the identifier of the server and the name of the new user, as well as further parameters for the creation of credentials. First, the authenticator of B creates a new credential and sends the public key pub_B back to S. During this process, the user is asked to confirm registration on device B. Next, the server S randomly generates a nonce n, links it to the username, and sends it to B. Then, n is transferred from B to A (e.g., using a

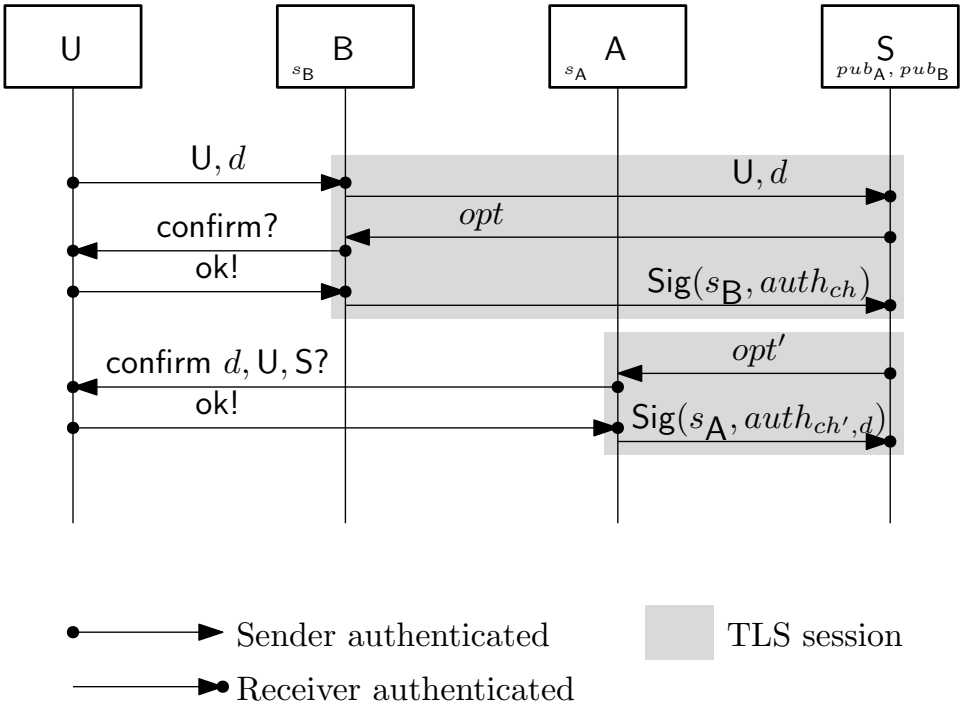


Figure 4.4: Sequence diagram for transactions in our protocol FIDO2D. Here, a user U authenticates a transaction with data d with a server S by using devices B and A. The devices B and A have a secret key s_B and s_A, and the server S knows the corresponding public keys pub_B and pub_A. The variables $auth_{ch}$ and $auth_{ch',d}$ contain ch and ch', d respectively and are further explained in the text.

QR code). Again, the user is asked to confirm the registration on the device A. Then, A can initiate a FIDO2 registration ceremony with S. Because A supplies n, S can link both registration ceremonies to the same username.

As a result of the registration, S is aware of public keys pub_A, pub_B linked to user U. The devices A and B know the secret keys s_A, s_B for the public keys pub_A, pub_B, respectively.

4.4.3 Transactions

The message flow during a transaction is shown in Figure 4.4. Again, the communication between the devices B and A and the service S is protected using a TLS session with server authentication. Recall that a user U wants to issue a transaction with data d at a server S. First, the user U enters the intended transaction data d on device B, which transfers the transaction data together with the username to S. S sends a set of options opt, containing a random challenge ch, to B. To complete the transaction initiation, B

must respond with a signature covering ch and S using the private key s_{B}. Before this data is signed by B's authenticator and the signature is sent to S, according to the FIDO2 protocol the authenticator has to confirm *user presence*, i.e., the user has to touch a button. In addition, because we require *user verification*, the authenticator has to authenticate the user, e.g., using a fingerprint scanner or a PIN. However, in this step U does not have to verify the transaction data as this was implicitly done by entering the transaction data on B.

The authenticator computes the signature over a hash of the servers' domain, flags, the authenticator's signature counter (which is used for clone detection), extension data, and a hash of the client data that contains the challenge ch. This follows the FIDO2 standard. In Figure 4.4 this is simply presented as $\mathsf{Sig}(s_{\mathsf{B}}, auth_{ch})$ where $auth_{ch}$ contains the aforementioned data. After receiving the signature from B, the transaction is not yet fully authenticated. The server S requires confirmation by A too. For this, the FIDO2 authentication protocol is performed between A and S. Note that in this step a fresh challenge ch' and the public key pub_{A} is used. S links both challenges ch and ch' to the transaction data d and user U.

A receives the transaction data d together with the challenge ch' as part of options opt' from S, and presents the transaction data, identifier of S and username U to the user. The user has to verify that this data is correct and decline the dialog otherwise. As before, the authenticator has to confirm *user presence* and *user verification*. For device A we use the Simple Transaction Authorization Extension (txAuthSimple) of FIDO2. This extension ensures that d is signed together with ch' and S. Thus, after U's confirmation, A not only signs ch' and S, but d too. Thereby the challenge ch' is linked to the transaction data d, which is important when the data is transmitted over an insecure channel. As we propose to use TLS, transferring d along the signature would suffice in our attack model. However, using the txAuthSimple extension allows adapting the protocol to devices that can only communicate insecurely with S. For example, a device A that is not connected to the Internet directly could transmit data to B to relay them to S. As the channel between A and B might be constrained, e.g., when transferring data manually with QR codes, using TLS is not reasonable in this scenario.

In our diagram, we denote the signature generated by B's authenticator as $\mathsf{Sig}(s_{\mathsf{B}}, auth_{ch})$, where $auth_{ch',d}$ depicts the data that is signed analogous to $auth_{ch}$ described before. S accepts the transaction with data d when both authentication ceremonies are successful, i.e., both signatures can be verified using the stored public keys and the signatures cover the correct challenges ch and ch'. In addition, the signature from device A must cover the correct transaction data d.

Figure 4.5: Screenshots of a user's view in our app prototype during registration. *Left:* Registration in the browser using FIDO2. *Right:* Registration in the app by scanning a QR code.

4.4.4 Prototypical Implementation

To verify the feasibility of our approach, we implemented a prototype of FIDO2D consisting of a server component for S and an Android app running on A. On the server, we use a Go library for FIDO2 by Duo-Labs[2]. We had to add support for the txAuthSimple extension. More specifically, the server has to check that the authenticator signed the extension data containing the transaction details and that they have not been tampered with. For our app, we use an Android library by Duo-Labs[3] to create credentials and sign supplied challenges. Again, we added support for the txAuthSimple extension, which mainly required signing the extension data. On device B, we use the Web Authentication (WebAuthn) browser API which is part of the FIDO2 standard and is supported by all modern browsers.

Following the principle (2DA.4) we limit access to the devices by a local authentication mechanism such as a lockscreen. On A we force the use of a lockscreen by enabling the option *setUnlockedDeviceRequired* of the Android Keystore[4]. This ensures that the private key can only be used if the device is unlocked. Access to the private key can be further restricted by requiring to push a physical button or displaying transaction data on a protected screen [124]. We assume that device B has a lockscreen set up

[2] `https://github.com/duo-labs/webauthn`
[3] `https://github.com/duo-labs/android-webauthn-authenticator`
[4] `https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec.Builder`

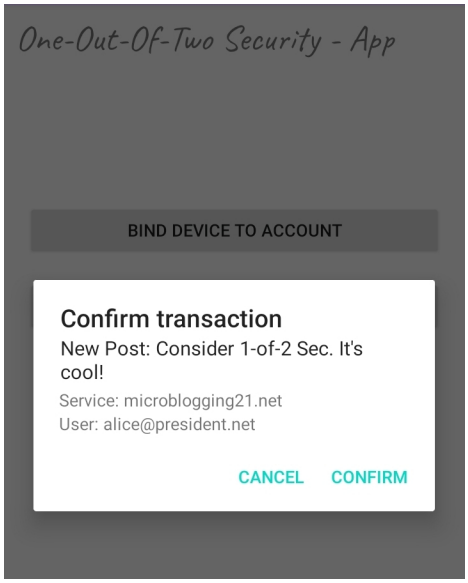


Figure 4.6: Screenshot of our app prototype during transaction confirmation in a fictitious micro-blogging scenario.

too. Furthermore, we enable the *requireUserVerification* parameter in the WebAuthn API and require the *User Verified* flag to be set in the signed authenticator data. This ensures that roaming authenticators that are prone to theft cannot be used without authentication, e.g., using a PIN.

Figure 4.5 shows the user interface of the registration process. On B the browser shows a pop-up asking the user to confirm registration with FIDO2. Afterwards, the user links the app on A to the account by scanning a QR code from the browser. In the background, the app establishes a connection to the server and initiates a FIDO2 registration ceremony as well. Figure 4.6 shows the transaction confirmation screen of our app. In this example, the user initiated a new post on a fictitious micro-blogging service. The app displays all relevant information including the username and identifier of the service.

Even though the Client to Authenticator Protocol (CTAP) is designed to relay data to an authenticator, we decided against using it for our app. CTAP requires the user to connect the smartphone to the computer during authentication [69]. However, Bluetooth and NFC are not available on all platforms [112] and USB requires a wired connection. Thus, we suggest using push messages and TLS for the communication of A and S. In our implementation, we use the push service Firebase Cloud Messaging[5]. However, we do not transmit any data in the push message, but request the app to establish a TLS connection to the server and retrieve data from there.

[5] `https://firebase.google.com`

Integration of a push-based transport into CTAP would be desirable, as it would allow FIDO2D to rely on standard FIDO2 only. Recently, attempts to integrate such a mechanism into CTAP [128] have been made, however, they are not yet included in the standardized protocol. We hope that our work strengthens these developments. Furthermore, our prototypical app could be superseded by integrated support for FIDO2 in mobile operating systems. This requires that the smartphone can be used as a roaming authenticator on another device and that it supports the txAuthSimple extension such that transaction data can be verified. However, the extension txAuthSimple has been removed in the second level of the WebAuthn standard [164]. This was justified by the fact that the extension was not implemented in any browser[6].

Although we do not consider recovery in our implementation, promising methods for recovery of FIDO2 authenticators have been proposed that do not sacrifice security by incorporating a backup authenticator [73].

4.5 Security Proof

We formally prove that our protocol FIDO2D fulfills the security notion *one-out-of-two-security* (see Section 4.3.3) with the help of the protocol verification tool *Tamarin Prover (Tamarin)* [125]. Our lemma definitions, which from Tamarin's point of view constitute the protocol's security properties, are inspired by [115]. We also analyze the effect of users not verifying transaction data on the security of FIDO2D. While our scheme does not fulfill *one-out-of-two-security* in this scenario, we prove that it still protects against most types of attacks including phishing and malware affecting the additional device.

In the following, we describe how we modeled our protocol as well as the security notion *one-out-of-two security*.

4.5.1 Our Tamarin Model for FIDO2D

Our model consists of 13 rules and two lemma definitions. We refer to Appendix B for a complete definition of each rule, including an enumeration of facts that are produced and consumed by it. Each rule's purpose is briefly explained below:

- new_server: Registers a new honest server as a global fact. Honest servers cannot be used to perform phishing attacks.
- register_first_device: Registers a new account at a server and connects a user's first device to it. The device is identified by a public key generated for the particular account. Account and device identities are then leaked to the attacker.

[6] `https://github.com/w3c/webauthn/issues/1386`

- register_second_device: Finishes registration of a user's account by connecting a second device to it. This step also leaks the public key registered by the second device to the attacker.
- init_transaction: A user commences a transaction at a server for which there already exists a personal account with two registered devices. After sending the transaction with its accompanying data to the network, the first device waits for a challenge from the server.
- receive_transaction: An honest server receives a transaction from a previously registered user. It generates a challenge and sends this back to the user's first device.
- phish_transaction: A user falls for a phishing attack and initiates a transaction on a phishing server.
- receive_transaction_phisher: A phishing server receives a transaction of a user that fell for a phishing attack.
- sign_nonce: A user's first device signs a challenge for a transaction that has also been started by her.
- verify_signature: A server verifies the signature of a previously issued challenge of a user's first device. It then generates a second challenge, which is sent to the user's second device.
- sign_second_nonce: A user's second device receives a transaction request. After the user verified that the displayed transaction data corresponds to the transaction she previously initiated, her second device signs the second challenge. The signature is sent back to the server.
- verify_second_signature: A signature of a user's second device is verified on the server. If verification succeeds the transaction is completed, i.e., the server executes the user requested transaction internally.
- compromise_first_device: A user's primary device is compromised. We model this by leaking the device's private key to the adversary.
- compromise_second_device: A user's additional device is compromised. We model this by leaking the device's private key to the adversary.

An example trace containing a transaction accepted by a server can be obtained by instantiating the rules in the order given in Figure 4.7, excluding rules that model phishing or compromise of devices. In our model, there are two rules that do not require custom facts to be present within the global state, namely new_server and register_first_device. Consequently, these two

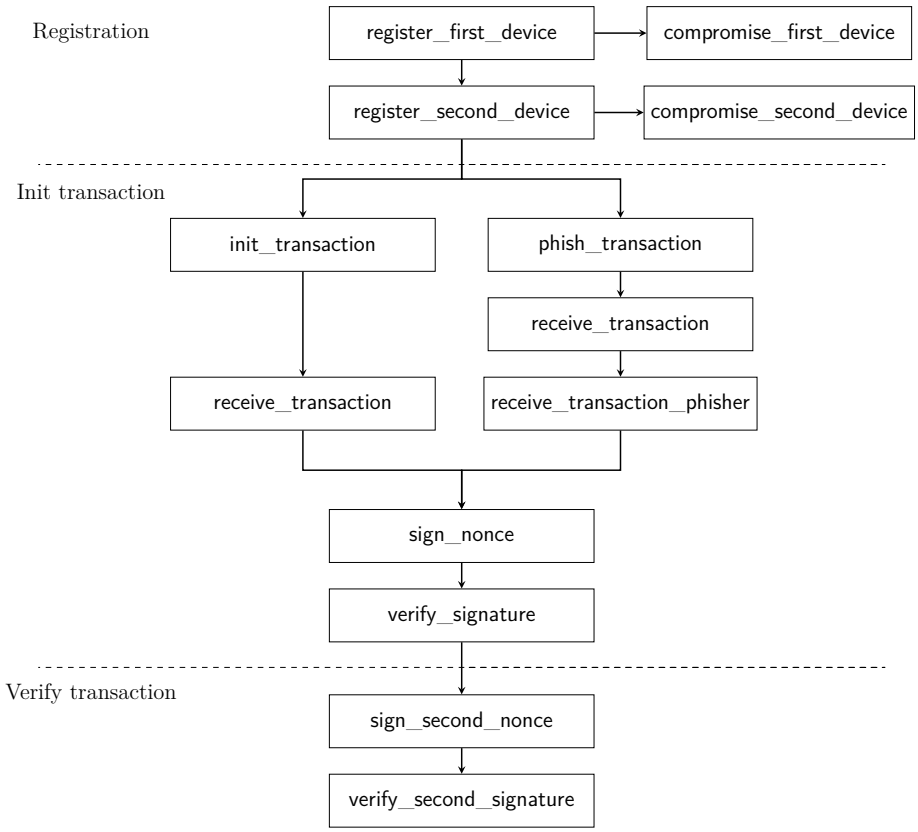


Figure 4.7: Structure of our Tamarin model. Pictured is the most straightforward order of instantiation of rules for normal and phishing transactions.

rules can be instantiated regardless of the current world state within Tamarin. In this example, protocol instantiation starts with registration of an honest server (omitted for clarity in Figure 4.7), registration of a user's devices at this server, initialization of a transaction and verification of a transaction by the server. These rules correspond to three distinct categories: *registration*, *transaction initialization* and *transaction verification*. In order to model phishing attacks, which is a transaction that is started on a malicious server, *transaction initialization* contains two more rules, init_transaction_phisher and receive_transaction_phisher.

4.5.2 How Our Model Captures Reality

Tamarin builds on the Dolev-Yao [54] attack model. In this model, the attacker obtains knowledge of every message sent over the network. He is also able to modify, suppress and inject messages, as well as re-send known messages at any time during protocol execution. However, the attacker is not able to break cryptographic schemes without knowing the key.

Although we expect typical implementations of FIDO2D to be embedded within already bootstrapped TLS sessions, our Tamarin model relies on unilateral authentic channels only. Specifically, we require the communication channels from the server to the user's devices to be authentic. We use rules suggested in the Tamarin manual to model authentic channels [154]. As a result, the actual payload of a transaction is always assumed to be transmitted in the clear over an attacker-controlled network. These simplifications allow us to present a more concise model in contrast to a model in which FIDO2 and TLS have also been realized. Furthermore, this highlights again that we assume a strong attack model. In a real-world implementation of FIDO2D, confidentiality is guaranteed by performing transactions within the context of a TLS session.

FIDO2D relies on FIDO2 and thus we also model a simplified version of the FIDO2 protocol. We use the built-in *signing* model of Tamarin that provides the necessary functions for a signature scheme. To model drawing a fresh nonce as challenge, we use the built-in fact `Fr(~nonce)`. We simplified the signed data to include the server identity and challenge only. Thus, in our model, the signature generation can be described as `sign(<S, nonce>, privkey)`. In our implementation, the second device signs the transaction details by usage of the FIDO2 txAuthSimple extension. Therefore, our model assumes that the authenticator data also contains the transaction data for the second device. This is expressed by the term `sign(<S, d, nonce>, privkey)`.

During registration, the registered public keys are written to the attacker-controlled network. We thereby model that authentication information stored on a server can be leaked by a data breach. However, we do assume the devices to be uncompromised during registration.

Our model considers phishing and malware attacks as described in Section 4.2. We model malware by explicitly allowing the attacker to compromise a device. This is depicted in the rules compromise_first_device and compromise_second_device by leaking the stored private key. In our protocol the private key is everything one needs to fully impersonate a device. Thus, it is not necessary to model other capabilities of malware such as the manipulation of transaction data explicitly.

To consider real-time phishing, we add the rules phish_transaction and receive_transaction_phisher. The rule phish_transaction allows the attacker to force the user to visit a potentially attacker-controlled website. In an actual attack, this could be accomplished by sending a phishing email with a link to the user. We assume that the user falls for the phishing attack and continues to follow the protocol by entering transaction data. The rule receive_transaction_phisher allows the attacker to answer the user's request with arbitrary transaction data and challenge. The whole protocol flow of a phishing attack can be found in Figure 4.7. After receiving transaction data from the user, the attacker initiates a manipulated transaction at the original

```
lemma only_user_initiated_transactions_accepted:

"All initiator transaction server #i. TransactionComplete
   (initiator, server, transaction) @i
==>
(
(Ex #j. TransactionBegin(initiator, server, transaction)
   @j) |
(Ex #k #l. CompromiseDev1(initiator, server) @k &
   CompromiseDev2(initiator, server) @l)
 )"
```

Listing 4.1: A FIDO2D security property which states that only honest (i.e., user initiated) transactions are accepted by a server, captured as a Tamarin lemma. In conjunction with *replay_attack_impossible*, this lemma constitutes *one-out-of-two-security*.

server. For example, this allows him to relay the challenge of a different transaction to the user's device.

4.5.3 Proving One-out-of-two Security

The security guarantees of our protocol are verified by Tamarin against two lemmas, *only_user_initiated_transactions_accepted* and *replay_attack_impossible*. Together they form our security notion *one-out-of-two-security*. The lemmas are inspired by the Tamarin proof of SecurePay [115] and analogous to the lemmas we defined for L-Pay in Section 3.5.4. We present the definition of the lemmas in Listing 4.1 and Listing 4.2 respectively.

The first lemma in Listing 4.1 states that every transaction that is accepted by a server must have been initiated by an honest user and has not been tampered with. However, this lemma is not sufficient on its own as it does not rule out replay attacks. In a protocol that is susceptible to replay attacks, an attacker may replay a legitimate transaction initiated by an honest user which would result in the same transaction being executed again. Thus, the second lemma shown in Listing 4.2 states that each accepted transaction must have been initiated by an honest user at a unique point of time. With these two lemmas we ensure that in a secure protocol, transactions by honest users can neither be started nor manipulated by an attacker.

Besides the targeted security guarantees, our security notion must also capture the attack model. The formalization of our protocol allows the attacker to compromise devices arbitrarily. However, to fulfill *one-out-of-two security*, a protocol must withstand the compromise of one of the user's devices but not both. Thus, we add a clause to ensure that the lemmas are not falsified when the attacker compromises both devices.

```
lemma replay_attack_impossible:

"All initiator1 transaction1 initiator2 transaction2
   server #i #j.

TransactionComplete(initiator1, server, transaction1) @i
   &
TransactionComplete(initiator2, server, transaction2) @j
   & not #i = #j
==>
((
Ex #k #l.
TransactionBegin(initiator1, server, transaction1) @k &
TransactionBegin(initiator2, server, transaction2) @l &
not #k = #l
) |
(Ex #m #n. CompromiseDev1(initiator1, server) @m &
   CompromiseDev2(initiator1, server) @n) |
(Ex #m #n. CompromiseDev1(initiator2, server) @m &
   CompromiseDev2(initiator2, server) @n) )
```

Listing 4.2: A FIDO2D security property which states that transactions cannot be replayed captured as a Tamarin lemma. In conjunction with *only_user_initiated_transactions_accepted*, this lemma constitutes *one-out-of-two-security*.

For simplicity, the lemmas do not impose any temporal order on the sequence of actions on the trace. Thereby, the lemmas are more general but consider traces with specific temporal order as well. For example, the point of time at which the start of a transaction is recorded should be before the corresponding completion of said transaction.

By using the Tamarin theorem prover, we verified that our protocol satisfies both lemmas *only_user_initiated_transactions_accepted* and *replay_attack_impossible* and thus exhibits one-out-of-two security. In the following, we sketch why the scheme intuitively fulfills one-out-of-two security and adheres to these lemmas. Assume a user's first device is compromised, and her second device is benign. Then, the attacker is able to initiate fraudulent transactions or manipulate benign transactions on the first device. During protocol execution the user is asked to confirm the transaction data $\hat{d}$ on her second device, although she did not initiate a transaction at all or initiated a transaction with data $d \neq \hat{d}$. Clearly, the user will not confirm this transaction and thus the server will not execute the transaction. On the other hand, assume her second device compromised and her first device benign. In this case, the attacker cannot initiate a transaction himself. If the user initiates a transaction, the attacker is not able to manipulate the

transaction data because the server links the challenge sent to the user's second device to the transaction data entered on the corresponding first device beforehand. Thus, the attacker can only confirm a transaction with transaction data that was chosen by the honest user.

4.5.4 Comparing Transaction Data

Our model expects the user to verify transaction data properly. User studies suggest that this is not always the case [94]. Furthermore, attackers have been found to successfully bypass authentication by triggering repeated push notifications to animate honest users into accepting them [107]. Similar to prior work, we extend our model to consider users that do not verify transaction data at all [12]. In our model, it is sufficient to remove parameters from the fact `UserWaitForConfirmation`. Thereby, the user only confirms a transaction after she initiated one but does not compare the transaction data. Under this assumption, our protocol does not satisfy *one-out-of-two-security*. An attacker can break the scheme by compromising the first device to initiate a malicious transaction. Then, the user confirms the transaction on the second device because she does not verify the transaction data. Below we describe the sequence of rules that leads to a successful attack.

1. new_server: A new honest server is registered.
2. register_first_device: The user registers her first device.
3. register_second_device: The user registers her second device.
4. compromise_first_device: The attacker compromises the first device of the user and leaks the private key sk stored on the device.
5. init_transaction: The user initiates a transaction with transaction data d.
6. receive_transaction: The adversary drops the message sent by the user's first device and initiates a transaction with transaction data d' instead. The service sends a challenge ch to the user's first device.
7. sign: The adversary signs the challenge ch with the stolen key sk and sends the signature to the service.
8. verify_signature: The service verifies the signature and sends the transaction data d' and a fresh challenge ch_2 to the user's second device.
9. sign_second_nonce: The user confirms the transaction without noticing that the transaction data differs. The second device signs d' and the challenge ch_2 and sends the signature to the service.

10. verify_second_signature: The service verifies the signature from the second device and confirms the transaction.

By removing the rule compromise_first_device we can prove that our protocol resists an attacker that compromises the additional device and carries out password and phishing attacks even if the user does not verify the transaction details at all. Thus, FIDO2D provides strong security guarantees and even provides *one-out-of-two-security* if the user verifies transaction data.

4.6 On When to Use FIDO2D

FIDO2D can improve security in several scenarios. Before we describe possible use cases, we present general guidelines and requirements.

4.6.1 Guidelines and Requirements

To adopt FIDO2D, the first step is to identify security-critical actions that should be protected from malware attacks. Our scheme can be used to achieve two separate security goals. First, the *confidentiality* of data stored on the server can be protected by requiring the user to confirm requests for said data. However, once the user approves a request, the data is shown on the primary device and might be eavesdropped by malware. Nonetheless, malware on a single device cannot get access to data protected by transactions without the user requesting or confirming it. Furthermore, transactions can protect the *integrity* of data stored on the server or activities triggered by the server. For example, money transfers are commonly protected by transactions as they change the account balance stored on the server. Actions have to be chosen carefully, as requiring *transaction authentication* for many actions can easily cause authentication fatigue, drastically reducing security benefits.

For the security of FIDO2D, it is important that transaction details contain all necessary information about the transaction, so that the user can verify them. As FIDO2D might be used for multiple accounts and services, the transaction details should always include the domain name of the service and the account name. Furthermore, the length of the transaction details should also be considered when identifying suitable actions.

Services can still provide a login mechanism and offer to initiate a transaction from the authenticated session. However, the login mechanism must resist password and phishing attacks and security-critical actions chosen by the provider must be verified by the user on the additional device. Of course, not all users want to confirm messages individually. Thus, the use of FIDO2D should be configurable. However, disabling FIDO2D for an account must also require verification with FIDO2D. Otherwise, FIDO2D can be bypassed trivially.

To be able to use FIDO2D, users need a smartphone and a computer that supports FIDO2 either with an integrated platform authenticator or with a roaming authenticator. Windows 10 provides a FIDO2 platform authenticator out-of-the-box[7]. Hence, many users only need to install an app on their smartphone. Our app can be used for multiple web applications that implement our protocol. While platform authenticators are bound to a specific device, a roaming authenticator might also be used to access an account from multiple devices.

4.6.2 Use Cases

FIDO2D is suited best for scenarios with security-conscious users as it unfolds it full potential when users verify transaction data properly. We introduce four use cases where FIDO2D can protect users against malware attacks.

Online Banking

Of course, FIDO2D can be used in online banking scenarios. For example, transactions can be used to protect the *integrity* of money transfers and settings such as transfer limits. For money transfers, the transaction details should include the recipient and the amount. Changing limits, should include the new value of the limit. These types of transactions can be verified on a smartphone properly.

Even though *transaction authentication* is already used in online banking, applying FIDO2D yields the following advantages. First, it advertises initiating and confirming transactions on a different device. And second, it replaces the use of passwords to login to the online banking system.

Electronic health record

In electronic health records, FIDO2D can be used to protect the *confidentiality* of stored health information. Sensitive information would only be accessible for a patient after confirmation on the smartphone. The transaction data should indicate which data was requested. Such requests can be verified on a smartphone properly.

Microblogging

In a microblogging service, the *integrity* of posted messages can be protected using FIDO2D. In this scenario, publishing a message might have a huge impact depending on the author. Messages of influential people such as the former president of the United States have been associated with stock market activity [79] but might also influence international affairs. As messages in a

[7] `https://fidoalliance.org/microsoft-achieves-fido2-certification-for-windows-hello/`

microblogging service are short (e.g., 280 characters on Twitter), they are suitable for verification on a second device. The transaction details should include the name of the account as well as the full message content.

Administration Panels

Management panels such as Plesk[8] allow users to carry out administrative tasks on remote servers. For example, DNS settings for registered domains can be configured. Manipulating the DNS settings of a domain allows to eavesdrop and manipulate all traffic destined to the domain [98]. Thus, protecting the *integrity* of the DNS settings using FIDO2D is beneficial. The transaction details should include the domain name, the record type and the value. Since this information is short, it is suitable for verification on a smartphone.

4.7 Related Work

In the following, we consider two types of related work: security models for web authentication and work that introduces web authentication schemes. Finally, we compare the security of existing web authentication schemes with FIDO2D.

4.7.1 Security Models

Jacomme et al. introduce a formal model for multi-factor authentication in the applied pi calculus [106]. They analyze the Universal Second Factor (U2F) protocol, a predecessor of FIDO2 and Google 2-step authentication. Their threat model includes phishing and malware attacks, as well as fingerprint spoofing to bypass RBA. Using ProVerif they identify several weaknesses of the analyzed protocols. Google 2-step is susceptible to phishing attacks that include fingerprint spoofing and U2F to malware controlling an attached authenticator. They propose that these protocols should incorporate verification of transaction data, which is exactly what FIDO2D provides.

Bonneau et al. [22] compare schemes proposed to replace passwords for web authentication regarding usability, deployability, and security. They consider a wide range of schemes such as password managers, federated schemes, OTPs, and hardware tokens. The security of a scheme is assessed based on the resilience to different kinds of attacks. However, phishing and malware attacks are not considered to their full extent. For example, malware is only assumed to steal credentials passively but not to manipulate the browser. Furthermore, real-time phishing attacks are excluded explicitly.

[8] https://www.plesk.com/

4.7.2 Web Authentication

FIDO2 is a standard for web authentication published by the FIDO Alliance. It consists of a widely implemented browser API called WebAuthn [164]. This API simplifies integration of strong authentication mechanisms in web applications. The underlying challenge-response protocol relies on public-key cryptography. Even though the protocol is resistant to real-time phishing, it does not prevent malware attacks [106].

This also applies to Google's Advanced Protection program [80]. It restricts login attempts to FIDO tokens; however, they can still be abused by malware. Other proprietary solutions such as Duo Push and Akamai MFA confirm login attempts using push messages [84, 58]. However, they do not authenticate transactions individually and are thus susceptible to malware using session hijacking or transaction manipulation (see Section 4.2).

The CAP protocol is a transaction authentication scheme used in online banking [56]. A dedicated card reader is used to allow the user to verify transaction details. Even though the CAP protocol is resilient to malware on the primary device, it does not exhibit one-out-of-two security. In the unlikely event that the card reader is compromised, the scheme can be attacked by using phishing to initiate a malicious transaction (see Section 4.2). Nonetheless, it provides a high level of security as compromise of the card reader is less likely than of a smartphone. However, the protocol is of proprietary nature and requires the user to carry an additional device, which has been shown to hinder adoption significantly [116].

Recent attempts to improve authentication schemes focus on banning weak schemes but do not reason about security properties and requirements systematically, e.g., the European Commission issued the revised Payment Services Directive (PSD2), a regulation requiring *strong customer authentication* for online banking [66]. PSD2 suggests that users have to be made aware of the transaction details, but does not prevent malware attacks, as it allows operating both factors on one device [95]. NIST introduced security levels for authentication called Authenticator Assurance Level (AAL) [82], but does not systematically consider what needs to be authenticated. Thus, these regulations do not solve the problems of two-factor authentication schemes.

Mannan and van Oorschot [123] introduce Mobile Password Authentication (MP-Auth). The scheme is based on a trusted smartphone that stores cryptographic keys. By incorporating public-key cryptography, the protocol achieves phishing-resistance. The authors use *transaction authentication* to thwart malware attacks, yet MP-Auth is only secure if the smartphone is not compromised. Because smartphones are multi-purpose devices that are connected to the Internet, they are susceptible to malware as well. Similarly, Chow et al. introduce a scheme that relies on a trusted smartphone [38]. The smartphone is used as a OTP generator that displays the transaction data and requires confirmation by the user.

Table 4.1: Security of transaction authentication schemes.

✓indicates that a scheme is secure in a given scenario. × indicates that this is not the case.
B is the device running the browser. A is an additional device.
Compare means that the user verifies transaction details. No Compare indicates that this is not the case.
The trust model lists all devices that must be uncompromised for the scheme to be secure.

	B compromised	*A compromised*	*B compromised*	*A compromised*	*Trust model*
	No Compare		Compare		
CAP reader [56]	×	×	✓	×	A
PhotoTAN [1]	×	×	✓	×	A
MP-Auth [123]	×	×	✓	×	A
Chow et al. [38]	×	×	✓	×	A
Secure Pay [115]	×	×	✓	×	TEE in A
FIDO2D	×	✓	✓	✓	A or B

Konoth et al. propose a solution to secure two-factor authentication when both factors are operated on a single smartphone [115]. Their scheme SecurePay uses OTPs and a trusted user interface to verify transaction data. The authors formally model and verify the security of their scheme using Tamarin [125]. By relying on a Trusted Execution Environment (TEE), Secure Pay is resilient to malware attacks. However, this is only true as long as attackers do not exploit vulnerabilities in the TEE implementation to escalate their privileges to the secure world, which has been successfully achieved in the past [32]. Furthermore, TEE implementations have been shown to be susceptible to fault injection and side-channel attacks [147, 139].

Similarly to our work, they identify the issue that the current way two-factor authentication is defined and used is insecure and give guidelines on how to improve upon that. However, they mainly focus on the fact that strict isolation of the two factors is needed which, even though it works in their case, in general is not sufficient to guard against malware attacks. In contrast, our work gives a universally applicable guideline for how to design secure authentication schemes, which are resilient to malware. In particular, our notion of one-out-of-two security does not require a special type of device and is agnostic to how isolation is achieved.

4.7.3 Security of Transaction Authentication Schemes

We summarize the security of web authentication schemes in Table 4.1. We only consider schemes that support *transaction authentication*, because other schemes are susceptible to malware attacks (see Section 4.3). Because users might not verify transaction data properly, we differentiate two scenarios: *compare* and *no compare* (see Section 4.5.4). Furthermore, we assume the attacker to either compromise B or A. In addition, the attacker may carry out password and phishing attacks (see Section 4.2). A scheme is considered secure in a given scenario, if the attacker is not able to execute a malicious transaction. As expected, none of the considered schemes provides security guarantees if both devices are compromised.

First, we analyze the security of the schemes when a user confirms transactions on an additional device without comparing transaction data. In this case, all considered schemes are vulnerable to malware on the primary device B because the user does not detect manipulation of transaction data. In contrast to other schemes, FIDO2D is secure if A is compromised (see Section 4.5). The reason for this is that transaction initiation is protected by FIDO2, which is resilient to password and phishing attacks.

If the user actually compares transaction data, all schemes protect against malware on the primary device B. Again, FIDO2D is the only scheme that is secure if A is compromised. Control of device A allows an attacker to tamper with transaction verification [95]. However, FIDO2D protects transaction initiation with FIDO2 against phishing and malware attacks. The other schemes rely on password authentication (potentially including RBA) that is susceptible to password attacks and real-time phishing. As described in Section 4.2, the attacker can mount a transaction initiation attack.

Thus, FIDO2D is the only scheme that fulfills one-out-of-two security. For users that verify transaction data, it provides resistance to password, phishing, and malware attacks as long as one device is not compromised. Even if the user does not verify transaction data at all, FIDO2D is resilient to malware on device A. Thus, it provides a higher level of security than existing schemes regardless of whether transaction data is verified or not.

4.8 Conclusion and Future Work

In this chapter, we showed how to protect security-critical web transactions against attacks with malware. Current web authentication schemes do not offer this protection.

We identified requirements for such authentication schemes, namely one-out-of-two security and transaction authentication. Web authentication schemes that fulfill these requirements protect security-critical transactions against malware attacks as long as one device is not compromised.

We introduced 2DA, a generic blueprint for designing web authentication

schemes that fulfill one-out-of-two security. Based on this blueprint, we designed and implemented a new web authentication scheme called FIDO2D, which is applicable to a wide range of use-cases. We proved the security of FIDO2D using Tamarin.

By relying on protocols and APIs of the FIDO2 standard, FIDO2D can be integrated into web applications easily. We demonstrate this by creating a prototypical implementation.

Chapter 5

Data Aggregation

The contents of this chapter are based on joint work with Florian Hahn and Florian Kerschbaum. The publications that previously appeared and contained parts of this chapter's content are listed below.

- Timon Hackenjos, Florian Hahn, and Florian Kerschbaum. "SAGMA: Secure aggregation grouped by multiple attributes". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 587–601. DOI: 10.1145/3318464.3380569 [90]

5.1 Introduction

Databases are ubiquitous in enterprise IT architectures. They store huge amounts of potentially sensitive data in a centralized place and are thus a lucrative target for attack. This threat is intensified by the current trend to outsource databases and analytics to third-party cloud providers. Attackers have already been identified to target database servers. For example, in 2022 security researchers found a piece of malware specifically crafted for Microsoft SQL servers [36].

Encryption at the client side can be used to mitigate those security threats. Standard encryption schemes, such as AES-GCM or RSA-OAEP, render database operations at the cloud provider impossible. Fully homomorphic encryption (FHE) enables the execution of arbitrary queries on encrypted data, but is currently still too inefficient for most practical use [3, 78]. Secure multi-party computation (MPC), e.g., SMCQL [15], requires multiple, mutually distrustful cloud providers increasing cost and administrative burden. Cloud providers can use trusted hardware modules, such as Intel's SGX [44, 167], but this only shifts the trust anchor from the cloud provider to the hardware manufacturer. A long history of bugs in the Intel Management Engine [65], bugs based on speculative execution [25], side-channel vulnerabilities [81], and architectural CPU bugs [23] create doubts in the security those hardware modules provide. A very active line of research addresses these

shortcomings by developing encryption schemes that enable query processing over encrypted data, e.g., [74, 88, 138, 135].

Kamara and Moataz [109] introduced the first system supporting a large class of SQL queries without using property-preserving encryption. However, their proposal lacks an efficient solution for data aggregation on the server side as they propose data aggregation on the client side after data filtering and decryption.

In this work, we bridge this gap and consider aggregation queries grouped by multiple attributes over encrypted data combined with filtering. Aggregation queries are the most common form of query in data analytics. For example, Piwik, a popular web analytics tool, which – among others – we use to evaluate our encryption scheme, determines the number of visitors of a site by country, browser, referrer, time, and many other attributes.

Designing a secure aggregation scheme is challenging, since the tuples composing a group are aggregated into one result, i.e., the access pattern of an aggregation query reveals the frequency of each group. Currently, there exist two approaches in encryption schemes for secure aggregation. First, *revealing the access pattern*: Deterministic encryption, e.g., used in CryptDB [138] allows grouping on ciphertexts in combination with partial homomorphic encryption for data aggregation. This approach allows grouping by any combination of attributes, but leaks the frequency of each individual grouping attribute value. Simple, yet efficient leakage-abuse attacks exploit this frequency information to reconstruct plaintext values with high accuracy as recent work demonstrates [130, 86]. Second, *pre-computing the groups into an index*: Seabed [135] thwarts frequency analysis attacks by flattening the histogram of attribute values. It uses multiple columns per attribute and inserts dummy tuples. However, all group values and possible combinations of grouping attributes in queries must be known at encryption time and pre-computed to be outsourced; thus, increasing server storage requirements exponentially. Further, filtering clauses (SQL's `WHERE`) must be mostly handled at the client.

In Section 2.6, we introduced Secure Aggregation Grouped by Multiple Attributes (SAGMA), a protocol for secure data aggregation that neither leaks the frequencies of individual grouping attribute values nor requires the combinations of grouping attributes to be known at encryption time. In this chapter, we evaluate the practical applicability of SAGMA to protect the confidentiality of stored data even when the database server is infected. SAGMA only reveals semantically secure ciphertexts and hides the frequencies of individual grouping attribute values during query execution by splitting the value domain into disjoint buckets and processing each bucket at once. Queries can contain any combination of at most t grouping attributes where t is a threshold chosen at setup time. SAGMA supports additional filtering clauses without relying on property-preserving encryption while significantly reducing the storage requirements compared to previous solutions.

5.2 Practical Applicability

In this section, we analyze the practical applicability of SAGMA to protect real-world database systems from malware attacks and propose improvements. More specifically, we study the resistance of the protocol to malware, and investigate if the performance and storage requirements of SAGMA are suitable for practical use.

5.2.1 Malware Resistance

SAGMA only reveals semantically secure ciphertexts to the database server. Thus, malware running on the database server only has access to encrypted data. However, during query execution SAGMA leaks additional information. In Section 2.6.4, we presented a simulation-based security proof for SAGMA with precisely defined leakage. The underlying security notion assumes a persistent, honest-but-curious attacker. That means, the attacker can monitor database operations passively to gain knowledge about plaintext data. However, it does not consider an attacker that actively interferes with the protocol. As homomorphic encryption schemes are by definition malleable, malware could manipulate stored ciphertexts and thereby change the result of a query. However, in the following we focus to protect against an honest-but-curious attacker and note that active security is still an open problem in data aggregation. As verified by the formal proof, during query execution, SAGMA only leaks the mapping of database rows to the group attribute buckets. The impact of this leakage can be reduced by increasing the bucket size. However, as we showed in Section 2.6.5 the bucket size cannot be increased arbitrarily without a performance penalty. Thus, in the following, we analyze additional mechanisms to reduce the leakage.

In a real-world scenario, the impact of the leakage largely depends on the distribution of the plaintext data. Rows that are part of different buckets can be distinguished using the access pattern; however, the underlying group values for each row cannot be extracted from this leakage. More precisely, group values that are mapped into the same bucket are indistinguishable for any adversary; an adversary distinguishing these rows with non-negligible probability could break the SWHE scheme, i.e., could distinguish encrypted monomials. Given a pseudorandom function that maps individual group values to buckets, an adversary cannot tell which group value is mapped into which bucket for group values with the same frequency. The provided security heavily depends on the characteristics of the underlying plaintext values – we emphasize, that this is true for all SSE schemes.

As highlighted by the designers of Seabed [135], the best security can be achieved by adding dummy rows to each bucket until the same frequency t is reached for each bucket hence making all buckets indistinguishable. However, a naïve implementation of such dummy rows would induce high

computational overhead. Specifically, assuming bucket size B and the B most-frequent group values occur r times in total then t is set to r. Assuming all other group values occur exactly once, each bucket is filled up with $t - B$ dummy values. The number of buckets is bounded by $\lceil \frac{|D|}{B} \rceil$, thus the number of total dummy values is bounded by $(\lceil \frac{|D|}{B} \rceil - 1) \cdot (t - B)$. In addition to dummy values, we describe additional mechanisms, such as optimizing the mapping function and splitting group attribute values in the next section. These can be used to minimize the total overhead for a targeted security level. We emphasize, that these protection mechanisms are data-dependent and their effect should be analyzed for each use-case individually.

Bucket Partitioning

The choice of the mapping function directly influences the bucket distribution and hence information leakage. For instance, consider bucket size $B = 2$ and three group values g_1, g_2 and g_3 with frequencies $1, 2$ and 3. Mapping g_1 and g_3 into the same bucket results in bucket frequency 4 and g_2 is mapped into a separate bucket with bucket frequency 2. An attacker aware of the group attribute distribution can reconstruct the mapping since the bucket frequencies are unique, i.e., they can only be generated by one mapping. In contrast, mapping g_1 and g_2 into one bucket and g_3 into a separate one, results in two buckets with the same frequency. Thus, the indistinguishability of values can be extended beyond values of the same bucket by proper partitioning.

Ceselli et al. [33] examine the use of hash functions for partitioning. They examine selection queries and introduce the exposure coefficient, which quantifies the average probability of correctly guessing the attribute value of an encrypted row based on auxiliary information. The exposure coefficient depends on several factors, e.g., the number of mapping functions resulting in the same bucket distribution, the number of attribute values and the number of buckets with the same frequency, the individual value distribution inside each bucket. In order to reduce exposure, they propose a larger collision factor of the hash function corresponding to larger bucket sizes in our scheme. Based on the work of Ceselli et al. we propose the following additional security mechanisms that increase security compared to deterministic encryption.

1. Optimal choice of the mapping function, i.e., bucket partitioning with minimal exposure.

2. Supplementing dummy rows and thus changing the number of values in buckets.

3. Partition group attribute values, e.g., split group attribute value g with high frequency in two distinct values $g.1$ and $g.2$.

Application	Grouping attributes		
	1	≤ 2	≤ 3
Nextcloud	100 %	100 %	100 %
Wordpress	97 %	99 %	100 %
Piwik	25 %	83 %	95 %

Table 5.1: Grouping queries with attribute numbers.

Optimal mapping function We choose a bucket distribution that minimizes exposure. Note, that this approach is limited by the plaintext distribution and the bucket size. For example, given bucket size 2 and three group values g_1, g_2 and g_3 that occur 1, 2 and 4 times respectively, all possible bucket partitions are distinguishable. Here, a specific choice of the mapping function does not introduce a large computational overhead and can be combined with dummy values and attribute value splits as further protection mechanisms.

Supplementing dummy values The structure of individual buckets can be altered by adding dummy values. Particularly, the number of elements in one bucket can be increased by adding dummy values containing value attributes all set to zero into specific buckets. Thus, they do not influence the aggregation result, but hide the distribution of attribute values. While inserting dummy rows for the use case of Ceselli et al. raises additional client overhead to filter rows, it only increases the server overhead in our use case.

Attribute value splits A group attribute value g can be split into two values $g.1$ and $g.2$ and thereby the frequency can be split into two summands. This requires the client to aggregate the sum for $g.1$ and $g.2$ after decryption but modifies value distribution in buckets, hence increase security.

5.2.2 Real-World Database Queries

As described in Sections 2.6.4 and 2.6.5, the storage requirements and performance of SAGMA mainly depend on the total number of grouping attributes t that are allowed in one query. However, this does not limit the applicability of SAGMA to real-world systems severely. As summarized in Table 5.1, many real-world applications use queries with a small number of grouping attributes only as our analysis of popular applications including Nextcloud 12, WordPress 3.7, and Piwik 3.2 showed. To determine the queries used by these applications, we set up a MariaDB database logging all queries into a file and ran the test suites of the applications. We describe the results of this evaluation below. Nextcloud only uses aggregation queries that group by a single attribute. Similarly, 97% of the aggregation queries used by WordPress contain a single group attribute; the largest query contains

three grouping attributes. Finally, Piwik, being an analytics platform, uses grouping functionality extensively. We report that 95% of the aggregation queries contain three group attributes or less. The largest query contains five grouping attributes. This indicates that establishing an upper limit for the number of group attributes supported in a single query does not restrict the applicability of the protocol to real-world database applications.

5.2.3 Storage Requirements

Finally, we analyze the storage requirements of our solution SAGMA and compare it with the following two approaches:

(i) pre-computing all results as discussed in Section 2.6.1

(ii) Seabed [135], another proposal for secure aggregation

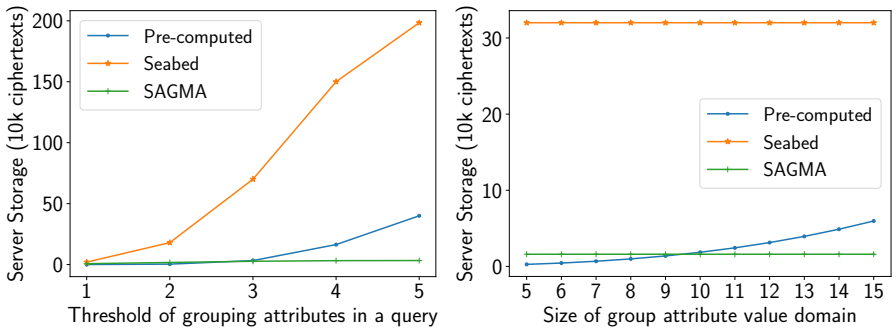
We analyze storage requirements on the server, as well as computation efforts required by the client. We assume a table with r rows, k value attributes, l grouping attributes and denote t as the maximum number of grouping attributes supported in one query. Further, we denote n as the number of filtering clauses queried by the client in combination with data aggregation, where the i-th filtering clause has a result set size of ρ_i. We refer to Table 2.9 for an overview of all variables used in the following analysis.

We assume the bucket size for SAGMA and the number of common values for Seabed to be equal for all grouping attributes denoted as B. For Seabed and SAGMA, we denote C as the number of aggregation results for a query with up to t grouping attributes, i.e., $C = |D|^t$. Originally, Seabed does not support grouping by multiple attributes, i.e., aggregating each attribute value combination individually. Thus, we assume all value combinations have been computed on the client and stored on the server.

Scheme	Server Storage	Client
Pre-computed	$\left(\sum_{i=1}^{t} \binom{l}{i} \cdot \lvert D\rvert^i\right) \cdot k \cdot n$	1
Seabed	$\left(\sum_{i=1}^{t} \binom{l}{i} \cdot ((B+1)^i - 1)\right) \cdot k \cdot r$	$\rho_i \cdot C$
SAGMA	$\left(\left(\sum_{i=1}^{t} \binom{l}{i} \cdot (B-1)^i\right) + k\right) \cdot r$	C

Table 5.2: Complexity of approaches hiding individual group attribute values. Notation is stated in Table 2.9.

In Table 5.2 we summarize the storage requirements on the server measured in ciphertexts and the required client computation effort for the different approaches. Naïve pre-computation requires each possible query result to be calculated, encrypted by the client and stored on the server. Thus, we



(a) Varying parameter t (b) Varying domain size $|D|$

Figure 5.1: Server storage requirements of the three schemes: Pre-computed, Seabed and SAGMA.

enumerate all possible grouping operations combining up to t attributes for storage calculation. The number of combinations for i grouping attributes has size $|D|^i$. Additionally, a different result is stored for each value attribute and for each possible filtering clause. Pre-computing and storing the results for all possible filtering clauses is impractical for real applications, e.g., assuming only one filtering column containing integer values and solely considering equality matching, there are already 2^{32} possible filtering clauses, increasing exponentially with additional value domains contained in further filtering columns.

In order to support grouping by multiple attributes with Seabed, $(B+1)^i - 1$ columns have to be stored for grouping by i grouping attributes. Again, we enumerate all grouping operations combining up to t attributes. Since Seabed encodes the value attribute in the group attribute column, all columns have to be stored for each value attribute separately. Seabed requires excessive client computation if combined with additional filtering clauses since ASHE is optimized for dense datasets. In the worst case, one operation has to be performed for each row contained in the filtering result for each of the C results. As a result, the client might have to perform more operations than there are rows in the database, making it less efficient than downloading the complete database and aggregating it locally on the client.

SAGMA requires $(B-1)^i$ monomials for a grouping operation with i attributes. As before, we enumerate all grouping operations combining up to t attributes. Value attributes and grouping attributes are stored separately and combined during aggregation. Hence, each value attribute is stored only once. SAGMA is interoperable with searchable encryption for filtering clauses, that is, filtering clauses are executed on the server and the resulting rows are subsequently aggregated using SAGMA. Client effort is minimal, as the number of different group value combinations is lower bounded by C and

for each combination the encrypted aggregation is decrypted by the client. SAGMA is the only scheme that scales with the number of value attributes and the number of possible filtering clauses at the same time.

We determine specific values for the server storage for $l = 4, t = 3, k = 2, r = 1000, n = 2$ and give the results in Figure 5.1. We evaluate different values for the maximum number of supported grouping attributes t in Figure 5.1a, and different group attribute domain sizes $|D|$ in Figure 5.1b. Seabed requires an excessive amount of storage space and SAGMA is superior to the pre-computed scheme for $t \geq 3$ and $|D| \geq 10$.

5.3 Related Work

Hacigümüş et al. introduce mechanisms to execute database queries on encrypted data by outsourcing a plaintext index [88]. They propose to partition an attribute's value domain into non-overlapping buckets and to outsource bucket identifiers instead of attribute values. This allows the database server to group rows according to the bucket identifiers even though the rest of the database row is encrypted. However, the scheme does not support server-side aggregation and thus the client has to aggregate the grouped values and refine the grouping after decryption. Therefore, most of the work has to be handled by the client. The authors do not provide a formal proof of security. Additional work on *bucketization* has been published by Hore et al. providing functionality for range queries [100, 99].

CryptDB provides an encrypted database system that supports a wide range of operations [138]. Aggregation queries with grouping are implemented using homomorphic encryption in combination with deterministic encryption. Deterministic encryption has been shown to be susceptible to inference attacks that can be used to recover plaintext data [130]. To reduce information leakage, attribute values are encrypted in multiple layers. However, during query execution, grouping attributes need to be deterministically encrypted and thus the client passes the encryption keys of the outer layers to the database server and enables inference attacks once again. Furthermore, CryptDB misses a formal proof of security. Other systems for encrypted databases refrain from using property-preserving encryption and leave server-side secure aggregation as an open problem [109].

Seabed combines additively symmetric homomorphic encryption (ASHE) and deterministic encryption to support aggregation queries with grouping [135]. ASHE has improved performance in comparison to asymmetric schemes such as Paillier, and is optimized for dense datasets, where multiple consecutive values are aggregated. This optimization, however, causes issues with additional filtering as expressed by `WHERE` clauses. In the worst case, the client's computation overhead for GroupBy statements in combination with `WHERE` clauses is the same as decryption and aggregation of all records

Scheme	Aggregation	Grouping	Security	Proof	Mult. Attributes
Bucketization [88]	×	×/✓	◐	×	×/✓
CryptDB [138]	✓	✓	○	×	✓
Seabed [135]	✓	✓	◐	✓	×
SAGMA[1]	✓	✓	●	✓	×
SAGMA	✓	✓	◐	✓	✓

Table 5.3: Comparison of our scheme SAGMA with related work.

matching the `WHERE` clause. Similar to the construction presented in Section 2.6.2, the authors encode the value attribute and the group attribute into a single plaintext and encrypt it using ASHE, hence they avoid deterministic encryption. Instead of using ciphertext packing, they create a new column for each group attribute value. To limit storage consumption, they propose an enhanced scheme that only introduces new columns for common group attribute values and one single column for less frequent ones. A column with deterministically encrypted group values is added to be able to aggregate the column that contains the uncommon values correctly. As this column only contains values in rows with uncommon group values, rows with common group values can be used to flatten the histogram by adding dummy values with no impact on the aggregation result. However, Seabed requires excessive storage space to support grouping operations with multiple attributes.

In Table 5.3, we compare all schemes mentioned in this section regarding their support for aggregation and grouping of multiple attributes on the server, classify their security properties, and whether they are formally proven secure. This shows that SAGMA offers a superior combination of functionality, flexibility, and provable security.

5.4 Conclusion

In this chapter, we analyzed the practical applicability of *SAGMA*, a protocol for secure data aggregation, to protect against a database server that is infected by malware. SAGMA hinders malware that leaks the database and monitors database operations from learning plaintext data. Previous constructions either leak more information or do not support grouping by multiple attributes together with additional filter clauses.

We propose mechanisms to further reduce the leakage such as optimizing the mapping function and supplementing dummy values. Furthermore, we analyze the storage requirements of SAGMA, and compare them with existing approaches. The choice of parameters, such as the bucket size and the supported number of grouping attributes, results in different trade-offs between security, supported queries, computational overhead and storage requirements. We analyze queries of real-world database applications such

[1] SAGMA without bucketization as described in Section 2.6.2.

as Nextcloud, Wordpress, and Piwik to verify that restricting the supported number of grouping attributes in a query does not limit the applicability of SAGMA to real-world systems.

Chapter 6

Related Work

In this chapter, we examine different types of related work. In Section 6.1 we review mechanisms that prevent infection of a system with malware, while in Section 6.2 we examine mechanisms used in cryptographic protocols to provide security guarantees even if a device is infected by malware. Existing security notions that consider malware are examined in Section 6.3. Finally, we provide an overview of malware-resistant protocols in Section 6.4.

This chapter can also be helpful for protocol designers to get a coarse overview of available mechanisms for malware-resistance. Furthermore, it allows to identify protocols with similar goals that might serve as a blueprint.

6.1 Preventing Malware Infection

Devices can be compromised in many different ways ranging from exploiting vulnerabilities in installed software over social-engineering attacks that trick the user into installing malware to tampering hardware - just to name a few [83, 157]. Thus, many lines of research emerged that explore ways to increase the security of devices and prevent them from getting compromised.

One line of research focuses on identifying vulnerabilities in hard- and software so that they can be fixed by the vendor. In vulnerability research, techniques such as static analysis and fuzzing are applied to discover vulnerabilities [122, 160]. Optimally, vulnerabilities are already identified during development. Therefore, tools and techniques are also integrated into software development processes to identify vulnerabilities. Furthermore, software verification can be used to prove that the behavior of a program conforms to a formal specification [2, 11]. However, this comes at a high cost and is thus not widely used [96].

To increase the difficulty of successfully exploiting a vulnerability, several exploit mitigation techniques have been introduced [136]. For example, Data Execution Prevention (DEP) ensures that certain memory regions that are intended to store data are not executable. However, exploit mitigations

have been bypassed multiple times in the past. For example, DEP can be bypassed by reusing code from the vulnerable program instead of writing the malicious code into memory.

To benefit from the work of security researchers that notify vendors of vulnerabilities in their products, it is important to regularly install security updates for devices [52]. Apart from updates, other security mechanisms should be enabled to strengthen the security of a device. For example, by setting up a lockscreen, access to a device can be restricted, using full-disk encryption stored information can be protected and secure boot impedes tampering [124, 70].

Many attacks rely on social engineering techniques to reach their goal. By tricking the user into clicking on a link that leads to a malicious website or to install malicious software, an attacker can get access to the user's system. Different types of security awareness training have been proposed to increase the resistance of users to social engineering attacks. However, awareness training techniques have to be chosen deliberately, as some popular techniques such as simulated phishing campaigns have been shown to potentially have an adverse effect [97, 159].

Antivirus (AV) and endpoint detection and response (EDR) software serve as a last resort to prevent infection of a system with malware. They apply different techniques such as signature-based detection where prominent byte patterns of known malware are used to identify samples on other systems and behavior-based techniques that focus on monitoring running software to identify malicious behavior [31]. However, AV and EDR software struggle to detect novel malware samples reliably as recent work confirmed [110].

While applying mechanisms to prevent devices from getting infected are important, they cannot rule out infection of a device entirely. Thus, we claim that infection of a device with malware should be part of each cryptographic protocol's threat model. As part of a defense in depth approach, compromising devices should be made more difficult, as well as mechanisms be used to make protocols malware-resistant.

6.2 Mechanisms for Malware-Resistance

Several mechanisms have been introduced in related work to restrict the impact of a malware infection. These mechanisms can be divided into two categories, mechanisms from the first category described in Section 6.2.1 are based on providing isolation of applications inside a device while mechanisms in the second category described in Section 6.2.2 rely on isolation between multiple devices. Mechanisms from both categories can also be combined to reach an even higher level of security.

6.2.1 Hardware-Based Isolation

When a device is infected with malware, this does not necessarily mean that the adversary has full control of the infected system. Modern computing platforms provide multiple barriers to restrict access to sensitive data or security-critical functionality using trusted hardware modules. Integrating such mechanisms in a cryptographic protocol can contribute to the security of the protocol regarding the resilience to malware infection.

Kernel Boundary

In modern operating systems regular applications run in user mode and are isolated from the operating system running in kernel mode. This isolation is implemented using different protection rings offered by modern processors. User-mode applications do not have permission to access hardware resources directly or manipulate the operating system. However, there are plenty of other opportunities for user-mode application to behave maliciously, e.g., by injecting code into other user-mode applications to dump sensitive information or manipulate applications. How extensive the capabilities of user-mode applications are, differs between operating systems and platforms.

In addition to manipulating other applications, user-mode malware can exploit vulnerabilities in the operating system to escalate privileges. Multiple malware samples have been found in the wild to use privilege escalation exploits [68]. Based on the extensive capabilities that user-mode malware has and considering privilege escalation exploits, the kernel boundary is not sufficient to restrict malware adequately.

Secure Key Storage

Cryptographic keys are an attractive target for adversaries because the security goals of cryptographic protocols can only be achieved if the keys stay secret. If keys are stored by the operating system or a user-mode application, they can be leaked by malware running with suitable privileges. Modern platforms provide ways to store cryptographic keys securely. Trusted Platform Modules (TPMs) as defined by the Trusted Computing Group (TCG) or GlobalPlatform Secure Elements (SEs) are specialized hardware components that provide security-related functionality such as *secure key storage*. Besides resistance to some forms of physical attacks, they also prevent malware running on an infected system to access stored keys.

A TPM (and a SE alike) provides an API to generate keys, import keys into the TPM, and execute cryptographic operations [6]. Most importantly, the API does not offer to export stored keys, at least not in plaintext. Thus, malware can interact with the TPM and request cryptographic operations but not extract keys without exploiting a vulnerability inside the TPM.

Typically, being able to use a key is sufficient for an adversary to attack a protocol. Nonetheless, storing a key inside a TPM, SE or similar hardware modules provides the following benefits:

(i) It prevents an adversary from relocating the key to another device and thus makes attacks more difficult.

(ii) It simplifies designing protocols that recover after a compromise which is formalized by the security notion Post-Compromise Security (PCS) (see Section 6.3.2 for more details).

(iii) It provides resilience to some forms of physical attacks.

Access to cryptographic operations can be further restricted through *user authentication* and *user presence* checks. User authentication with a PIN prevents malware from executing cryptographic operations without knowing the PIN. This protection is far from perfect since the PIN is commonly entered on the infected device and thus accessible to (high-privileged) malware.

User presence checks are intended to verify that a user is physically present. This is often implemented using hardware buttons that are wired to the hardware module and must be pressed by the user. Even though this check cannot be bypassed by malware, malware can still manipulate legitimate requests for cryptographic operations by the user and thereby execute arbitrary operations. However, *user presence* checks limit the number of operations that an adversary can execute and might support users in uncovering malicious behavior if legitimate requests fail frequently.

Trusted Execution Environment

Hardware modules that support a specific functionality such as *secure key storage* offer very limited extensibility. Apart from hardware modules that provide a specific functionality, solutions were designed to allow loading and running arbitrary software in a compartmentalized environment. A TEE as defined by GlobalPlatform is an execution environment that is isolated from the regular execution environment of a system [148]. In particular, this means that code running inside the TEE is protected from an adversary that compromised the operating system.

The GlobalPlatform definition does not dictate implementation details. Thus, a wide range of implementations that rely on hardware-based isolation such as Intel SGX, AMD PSP, and ARM TrustZone emerged [166]. Hardware resources can be shared between the regular execution environment and the TEE. This allows to build primitives like a Trusted User Interface (TUI) to display information that cannot be manipulated by malware [115, 104].

A disadvantage of using a TEE in a cryptographic protocol, is that it limits the applicability of the protocol. Since different TEE implementations

are not compatible with each other, supporting heterogeneous devices such as mobile devices, laptops, and desktop computers is expensive.

Furthermore, several attacks were discovered in recent years that question the level of security that TEEs provide. TEEs have been shown to be susceptible to side-channel and transient-execution attacks [81, 147, 25]. Recently, an architectural CPU bug has been discovered that allows to break the security of Intel SGX [23]. Cerdeira et al. review vulnerabilities found in TEE implementations based on ARM TrustZone. They note that these TEE systems expose a large attack surface and miss state-of-the-art protection mechanisms such as address space layout randomization (ASLR). Vulnerabilities do not only affect TEE implementations but were also found in many trusted applications running inside the TEE.

In addition to isolating security-critical software from malicious software running in the regular execution environment, according to GlobalPlatform TEEs should provide protection against some forms of physical attacks. Prior work determined that protections against these types of attacks is limited [139]. They were able to break security guarantees of ARM TrustZone using fault injection attacks.

Because of the ambiguous security guarantees and fading support [142] of TEEs, in this thesis we rely on the isolation of two separate devices instead. This approach is also described in more detail in Section 6.2.2. For a comparison of our approach with TEE-based systems in a specific use case, we refer to Section 4.7.2.

6.2.2 Distributed Computing

An alternative to achieving isolation inside a device, is to distribute an application that is part of a protocol between multiple devices. For example, in many online-banking protocols, a transaction is verified on a different device than it was initiated originally. Thereby, transactions can be executed securely, even if the primary device is compromised. To achieve this, most of the protocols introduce an additional device that is considered trusted [123, 56]. The problem of this approach is that it only provides limited protection against malware attacks. In the case that the trusted device is infected, the security guarantees are lost. Because general purpose devices such as smartphones provide a huge attack surface, some protocols rely on special-purpose hardware devices such as smart card readers or FIDO tokens (see Chapter 4). These devices are specifically built and hardened for their use case and are thus thought to be less prone to attack.

Many multi-party computation (MPC) protocols are specifically designed to withstand attacks where an attacker compromises a subset m of n parties (or devices) of a protocol (with $m < n$). These protocols allow multiple parties to jointly evaluate a function, without disclosing their individual inputs. To achieve this, MPC protocols apply a wide range of cryptographic primitives

such as commitments, secret sharing, oblivious transfer and homomorphic encryption [48, 137, 140]. However, use cases that benefit from these specific security guarantees are rare. In Yao's Millionaires' problem, two millionaires want to know which of them is richer [165]. This is a classic problem, that can be solved with MPC.

The security of our proposed protocols FIDO2D and L-Pay relies on the isolation of devices too. As our protocols target different security goals, namely authenticity and integrity of transaction data, they are not directly comparable to MPC. However, the attack model of our security notion *one-out-of-two security* is a special case of the aforementioned attack model of m compromised parties out of n parties for $m = 1$ and $n = 2$ and each party represents a device. *One-out-of-two security* guarantees that the protocol stays secure even if one of the two devices is compromised.

Cross-platform infection has been introduced in related work as a way to break authentication schemes relying on multiple devices [114, 53]. The authors exploit synchronization features to use control of one device to install malware remotely on another device that is synchronized with the first device. Thus, measures should be taken to isolate the two devices, such as disabling synchronization features. Furthermore, individual devices should be hardened to make compromise more difficult, e.g., by using security features such as lockscreens, full-disk encryption and secure boot. Using heterogeneous devices with different computing platforms and operating systems can also contribute to make infection of both devices more difficult, as vulnerabilities rarely affect multiple platforms.

In addition, mechanisms to determine the state of a remote system can be integrated into protocols to strengthen isolation. Remote attestation is a feature offered by hardware modules such as TPMs that allows an authorized party to verify the state of a system remotely [161]. TPMs provide multiple registers that contain hashed values representing the current system state. For example, during the boot process all executables and configuration data are hashed into these registers [85]. During a remote attestation protocol, these register values are transferred as part of a signed quote to the verifying party. Changes in the state of a remote system may indicate compromise of that system [71]. For example, malware modifying the boot configuration to achieve persistence also results in a different measurement that can be detected with remote attestation. These systems can then be excluded from the protocol to prevent further damage. In the FIDO2 registration ceremony, remote attestation can be used to restrict registration to approved authenticators that are in a valid state [164]. The security benefits of using remote attestation depends on the scope of the measured data. Measuring as much information as possible and including it in a quote would be beneficial to reveal more types of manipulation of the system. On the other hand, this could diminish the availability of the system as valid system changes can influence the measurements and get a system locked out from a protocol.

Sustaining operation of a system despite failure of one or multiple components is the goal of fault-tolerant systems. Besides software errors, faults can be triggered by adversaries to impair availability. In distributed systems, there might be inconsistent information about whether a component failed or not. Such a condition is also known as Byzantine fault. A common example of Byzantine faults is the Byzantine generals problem where multiple generals must agree on whether to attack the enemy or retreat [118]. This is especially challenging, because generals might behave maliciously and provide other generals with inconsistent information. As a remedy, Byzantine fault tolerance algorithms have been proposed to reach consensus, e.g., using state machine replication [30]. Consensus algorithms are used to provide different security goals such as availability and integrity. For example, the cryptocurrency Bitcoin uses a consensus algorithm to realize electronic payment without a trusted bank. The proof of work consensus algorithm is used to protect the integrity of a global transaction log. However, Bitcoin does not protect against the compromise of payment clients to create arbitrary transactions, which is the focus of Chapter 3.

6.3 Security Notions

A security notion captures both the targeted security guarantee, as well as the capabilities of the adversary [111]. In the following, we examine security notions that consider compromise of a device.

6.3.1 Perfect Forward Secrecy

Perfect forward secrecy (PFS) has been introduced in the context of authenticated key exchange (AKE) protocols [87]. This security notion states that exchanged session keys must stay secret, even if the long-term private keys of the protocol's participants are leaked later on [40]. For example, using public-key encryption to transmit a session key does not suffice to achieve PFS because an attacker that records the exchanged messages can decrypt the ciphertext once he learns the long-term private key.

PFS is not solely academic but also relevant for real-world systems. For example, the protocol TLS 1.3 achieves PFS [45]. Regarding infection by malware, PFS is not a very strong security notion, since it only states a guarantee for past protocol executions. Once a device is infected or rather the private key is leaked, current and future protocol executions do not fulfill session key secrecy. In this regard, PFS is weaker than *one-out-of-two security* which also protects protocol executions during and after a compromise.

6.3.2 Post-Compromise Security

PCS is another security notion for AKE protocols that states security guarantees for protocol executions after a compromise [40]. The authors differentiate between *weak compromise* and *full compromise*. While the first grants the adversary temporary control of cryptographic operations with the long-term private key, the latter allows the adversary to learn the long-term private key itself. There are two forms of PCS depending on the particular adversary model. The first form assumes that an adversary only achieves *weak compromise*, e.g., because the long-term private key is stored inside a hardware security module (HSM) such as a TPM. To fulfill this form of PCS, a protocol must ensure freshness of protocol messages to prevent precomputation. The second form allows for a *full compromise*; however, it assumes that an uncompromised session takes place after the compromise. For example, if an adversary infects a device and steals the private key but is not able to persist access to the device, protocols fulfilling PCS can provide security guarantees for future protocol executions.

The Signal protocol, which is already used in real-world messaging apps, can achieve PCS when implemented correctly [41]. However, Cremers et al. point out that the implementation of the Signal protocol as it is used in many messaging apps including Signal, WhatsApp, and Facebook does not fulfill PCS [46]. The authors implement an attack that involves cloning a device and impersonating the user afterwards. This attack should be prevented due to PCS, however, the experiment showed that the messaging apps are vulnerable nonetheless. PCS has mainly been analyzed in regard to message secrecy, however, it is also applicable to authenticity [55].

PCS is a relevant security notion to provide malware-resistance. However, PCS does not promise security guarantees while a device is compromised but only once the infection is detected and the device cleaned up or the attacker loses access otherwise. Our proposed security notion *one-out-of-two security* does even provide security guarantees during the compromise.

6.4 Conclusion

Various protocols were introduced in related work that provide some form of malware resistance. They differ in many aspects such as the desired functionality, the targeted security goal, the used approach and the proof technique. We give an overview of the protocols that we deem most relevant in Table 6.1.

We examine the targeted security goal such as authenticity, integrity and confidentiality. As in most protocols authenticity and integrity co-occur, we do not differentiate them. We also consider the targeted functionality by mapping the protocols to one of our use cases. However, two protocols target another use case namely instant messaging.

Protocol	*Reference*	*Security goal*	*Use case*	*Category*	*Attack model*	*Infected device*	*Proof technique*
L-Pay*	CH 3	A/I	EP	DC	1-of-2	CL	UC, TMN
VR-mobileCash	[141]	A/I	EP	DC	TD	CL	-
FIDO2D*	CH 4	A/I	WA	DC	1-of-2	CL	TMN
SecurePay	[115]	A/I	WA	HB	TH	CL	TMN
CAP reader	[56]	A/I	WA	DC	TD	CL	-
PhotoTAN	[1]	A/I	WA	DC	TD	CL	-
MP-Auth	[123]	A/I	WA	DC	TD	CL	PCL
Chow et al.	[38]	A/I	WA	DC	TD	CL	-
FIDO2 Tokens	[8]	A/I	WA	DC	TD	CL	GB
FIDO2 Platform	[164]	A/I	WA	HB	TH	CL	-
MoDUSA	[55]	A/I	IM	DC	PC	CL, SRV	GB
SAGMA*	CH 5	C	DA	DC	HC	SRV	SB
CryptDB	[138]	C	DA	DC	HC	SRV	-
Seabed	[135]	C	DA	DC	HC	SRV	SB
SMCQL	[15]	C	DA	DC	HC	SRV	-
Double-ratchet	[41]	C	IM	DC	PC	CL, SRV	GB

Table 6.1: Overview of malware-resistant protocols.

The abbreviations used in the table are described below:

*	Protocol introduced in this work
CH	Chapter
A/I	Authenticity/Integrity
C	Confidentiality
EP	Electronic payment
WA	Web authentication
IM	Instant messaging
DA	Data aggregation
HB	Hardware-based isolation
DC	Distributed computing
1-of-2	One of two devices is compromised
TH	Trusted hardware module
TD	Trusted device
PC	Post compromise
HC	Honest-but-curious
CL	Client
SRV	Server
UC	Universal Composability
TMN	Tamarin
PCL	Protocol Composition Logic
GB	Game-based security
SB	Simulation-based security

We provide a rough categorization of the used mechanisms. More specifically, we examine whether they rely on hardware-based isolation in a device or isolation between devices. Furthermore, we differentiate the underlying attack model. Some protocols assume a trusted device, a trusted hardware module, an attacker that is passive (honest-but-curious), an attacker that loses access to a device after infection (post compromise) or one of two devices to not be compromised (1-of-2). In addition, we denote which type of device is thought to be infected. Because all analyzed protocols use client-server communication, either the client or the server is considered infected. And finally, we investigate if the protocol provides provable security and if so, which proof technique is used.

The security goals of the protocols in Table 6.1 are related to the respective use cases. The use cases electronic payment and web authentication target authenticity (integrity), in the data aggregation use case confidentiality is the targeted security goal. In contrast, instant messaging targets both authenticity (integrity) and confidentiality of the message contents. Most protocols rely on distributed computing to achieve malware resistance. However, there are also two protocols that use hardware-based isolation: SecurePay and FIDO2 platform authenticators.

Taking a look at the attack model, we notice that the use of trusted devices is popular in electronic payment and web authentication. Two protocols rely on trusted hardware, and our own protocols L-Pay and FIDO2D are based on one of two devices being uncompromised. In instant messaging, the attack model resembles a post-compromise scenario where a device is compromised but cleaned afterwards. In data aggregation, an honest-but-curious attacker is assumed. This attacker operates passively, follows the protocol and tries to learn as much information as possible. An active attacker that deviates from the protocol could affect the correctness of the aggregation result.

In most protocols, the client device of a client-server architecture is assumed to be compromised. This applies to the electronic payment and web authentication use cases. In contrast, the server is assumed to be compromised in the data aggregation scenario. Furthermore, in instant messaging the server is considered compromised and according to the post-compromise scenario, a client can be compromised for a limited time.

Regarding provable security, more than half of the protocols provide a formal proof. The used proof techniques vary widely in the electronic payment and web authentication use cases. However, in the instant messaging and data aggregation use cases, game-based proofs and simulation-based proofs are predominantly used respectively.

Our analysis of related work emphasizes that for several use cases cryptographic protocols that provide at least some form of malware resistance already exist. Nonetheless, this work takes a step forward in several aspects. We propose one-out-of-two security, a new security notion for cryptographic protocols that rely on distributed computing to achieve malware resistance.

Our protocols L-Pay and FIDO2D achieve one-out-of-two security and improve upon many existing protocols that rely on a single trusted device and offer an alternative to protocols that require trusted hardware modules. In addition, we provide formal models to verify that our protocols fulfill one-out-of-two security using Tamarin. For data aggregation, our protocol SAGMA provides an adjustable level of security that improves upon existing protocols. We validate the practical applicability of SAGMA by analyzing queries of real-world database applications.

In summary, our work paves the way for the adoption of cryptographic protocols with malware resistance in the real world. We provide guidelines, as well as protocols that can serve as a template for real-world implementation. Furthermore, with our protocols and formal models we prepare the ground for the design of malware-resistant protocols for other use cases.

Chapter 7

Conclusion

In this work, we addressed the susceptibility of cryptographic protocols that are used in the real world by millions of people every day to malware attacks. Malware is a catch-all term for programs that behave maliciously. It is used by attackers to log keystrokes, dump credentials, and manipulate software running on the system. This allows to bypass various security mechanisms that were never designed to protect against attackers on the local system. In this work, we were concerned about the consequences of the infection of a system with malware to the security guarantees of cryptographic protocols.

As shown in related work, many protocols used in real-world systems are susceptible to malware attacks [56, 93, 129, 130]. The main problem is that these real-world protocols lack the formal treatment that is usually used in academic protocol design and thus the targeted security guarantees are not precisely defined. Furthermore, during protocol design an attack model was assumed that has been shown to be too weak for today's threat landscape. An attack model describes the capabilities of an attacker. Most real-world protocols are based on the Dolev-Yao attack model, where the attacker controls the network between computer systems and is able to read and manipulate exchanged messages arbitrarily [54]. However, this model does not assume the attacker to compromise systems with malware and thus many protocols are vulnerable to this type of attack.

The goal of this thesis was to determine how cryptographic protocols can be designed to provide resilience to malware while still being applicable to the real world. For that purpose, we examined three use cases: electronic payment, web authentication, and data aggregation. In each use case, we adopted a similar series of steps. We started by analyzing the security of existing protocols for that use case and confirmed the results from related work that most currently deployed protocols do not resist malware attacks. In particular, even protocols using smart cards or relying on two-factor authentication are susceptible.

From these insights we derived requirements for malware resistance and based on these requirements, we designed new cryptographic protocols. We used formal methods to model our security notion and protocol. Finally, we verified the security guarantees using a formal proof of security.

We see our main contribution in the fact that we provide formalisms to precisely define and verify what it means for a protocol to be resilient to malware. In addition, we define guidelines to design secure protocols and provide new protocols for our use cases that could serve as a template for real-world adoption.

We introduced formal models and security notions to prove resilience of our protocols to malware attacks. In particular, we propose *one-out-of-two security*, a security notion for cryptographic protocols where a human user operates two devices. Protocols that fulfill *one-out-of-two security* must stay secure even if one of the two devices is compromised. We use Tamarin, a tool for the formal verification of cryptographic protocols to verify our claimed security guarantees. We formulated the main research question of this thesis as follows:

> *How can cryptographic protocols be designed for real-world systems to provide resilience to attacks with malware?*

In this work, we showed that designing cryptographic protocols with malware-resistance is indeed possible. We introduced L-Pay a protocol for secure electronic payment satisfying *one-out-of-two security*, FIDO2D a protocol based on the FIDO2 standard protecting transactions in the Web with *one-out-of-two security*, and SAGMA a protocol for secure data aggregation in encrypted databases with semantic security and precisely defined leakage.

Our protocol L-Pay improves upon EMV, the worldwide standard for electronic payment. EMV has been found to be susceptible to a plethora of attacks such as relay attacks, pre-play attacks, and man-in-the-middle attacks [13, 20, 57, 129]. One particular weakness is that compromising an automated teller machine (ATM) is sufficient to break the security of this protocol standard. L-Pay offers secure money withdrawal at an ATM or payment at the point of sale (POS). We achieved this by enabling the user to verify the transaction data on a smartphone. In contrast to previous protocols, L-Pay fulfills one-out-of-two security and as such also protects transactions when either the primary device or the smartphone is compromised. We formally verified this claim using Tamarin.

In our second use case, we analyzed authentication in the Web. Passwords are still the prevalent mechanism for authentication there. Two-factor authentication is on the rise; however, most web authentication schemes are inherently vulnerable to malware attacks. One reason for this is that web authentication schemes usually establish an authenticated session during login. This session can be hijacked by stealing cookies stored in the

browser using malware. Online banking schemes authenticate transactions individually instead of establishing a trusted session, nonetheless, most of them are vulnerable to malware attacks. We introduced 2DA, a blueprint for designing web authentication schemes with malware resistance and FIDO2D, a protocol for web authentication based on our blueprint. FIDO2D relies on the FIDO2 standard to securely initiate and confirm a transaction on two separate devices. We verified that our protocol fulfills one-out-of-two security using Tamarin and confirmed the feasibility of our approach by creating a prototypical implementation. Integration into the FIDO2 standard would pave the way for broad adoption of our approach. We urge the FIDO Alliance to fully embrace one-out-of-two security and transaction authentication as a design paradigm for secure web authentication in future versions of their standard.

Finally, we analyzed the resilience of database servers to malware attacks. Commonly, databases are not encrypted and as such can be accessed by malware that compromised the database server. Encrypting the data with a regular encryption scheme on a client protects the confidentiality of the data against malware on the server, however, at the same time it prevents processing the data. Special cryptographic schemes bridge this gap by supporting a limited set of database operations while protecting the confidentiality of the stored data. Data aggregation with grouping is a common functionality used in data analytics. Previously, a combination of deterministic encryption and homomorphic encryption was proposed to support these types of operations [138]. However, deterministic encryption leaks the frequencies of attribute values that can be exploited in leakage-abuse attacks [130]. We proposed SAGMA, a protocol for secure aggregation with grouping that provides a novel trade-off between security and performance by hiding the frequencies of individual attribute values using bucketization. SAGMA relies on homomorphic encryption in combination with searchable symmetric encryption (SSE). It provides semantic security and query execution only leaks the bucket membership of rows. By increasing the bucket size, the impact of the leakage can be reduced. We showed how the plaintext data can be preprocessed to further reduce the leakage and analyzed queries of real-world database applications to verify the applicability of SAGMA to real-world systems.

With our protocols, formal models, and security notions we have taken important steps towards the use of malware-resistant protocols in real-world systems. We showed that designing secure protocols that protect against malware is possible. In particular, we provide cryptographic protocols with malware-resistance for three use cases. Even though no one-size-fits-all approach for secure protocol design exists, we provided guidelines, as well as security notions and formal models that can serve as a foundation to design secure protocols for other use cases in the future.

Future Work Apart from the three use cases we examined in this thesis, many other real-world protocols are susceptible to malware too. For example, remote administration protocols such as Remote Desktop and SSH rely on session authentication and are thus vulnerable to session hijacking. Designing a malware-resistant protocol for remote administration is left for future work.

Usability aspects have a huge impact on the adoption of a protocol in practice. Thus, future research could conduct usability studies to examine how well users cope with our protocols. In particular, the verification of transaction details on a smartphone might be prone to human errors.

We proposed the notion one-out-of-two security to protect the authenticity and integrity of transaction data. It is still an open problem how the confidentiality of data requested by an honest user can be preserved when one device is infected by malware. Once data is displayed on the screen of a device, it is accessible to malware. Thus, a secure scheme would have to split the information between devices and enable the user to combine them.

Determining which actions of a web application should be protected by transaction authentication is challenging. There is a fine line between increasing security by covering more actions and causing authentication fatigue diminishing security ultimately. Furthermore, one has to ensure that actions that are not protected by transaction authentication cannot impact state that should only be modifiable by actions protected by transaction authentication. Otherwise, transaction authentication can easily be bypassed. Providing a framework to determine suitable actions for transaction authentication and eliminating the possibility of privilege escalation using unprotected actions is left for future research.

Appendix A

Tamarin Model for L-Pay

The Tamarin model for our protocol L-Pay from Chapter 3 can be found below. This also includes the formalization of our security notion *one-out-of-two security* for electronic payment.

```
theory lpay
begin
builtins: asymmetric-encryption

rule create_bank:
    [
    ]
    --[BankCreated($B)]->
    [
        !Bank($B),
        Out($B)
    ]

rule create_atm:
    [
    ]
    --[ATMCreated($ATM)]->
    [
        !ATM($ATM),
        Out($ATM)
    ]

rule init_app:
    [
        Fr(~k_app)
    ]
    --[]->
    [
        !App_Pk(pk(~k_app)),
        !App_Ltk(pk(~k_app), ~k_app),
        UnlinkedApp(pk(~k_app))
```

```
32      ]
33
34  rule register_account:
35      [
36          !Bank(B),
37          !App_Pk(pk_app),
38          UnlinkedApp(pk_app),
39          Fr(~pin)
40      ]
41      --[AccountRegistered($Acc, B)]->
42      [
43          !Account($Acc, B, pk_app, ~pin),
44          Out(<$Acc, pk_app>)
45      ]
46
47  rule init_transaction:
48      [
49          !Bank(B),
50          !ATM(ATM)
51      ]
52      --[TransactionInitiated($Acc, B, ATM, $d)]->
53      [
54          UserWaitForConfirmation($Acc, B, ATM, $d),
55          ATMTransactionStarted($Acc, B, ATM, $d),
56          Out(<$Acc, B, ATM, $d>)
57      ]
58
59  rule attacker_init_transaction:
60      [
61          !Bank(B),
62          !ATM(ATM)
63      ]
64      --[MaliciousTransactionInitiated($Acc, B, ATM, $d)]->
65      [
66          AttackerWaitForConfirmation($Acc, B, ATM, $d),
67          ATMTransactionStarted($Acc, B, ATM, $d),
68          Out(<$Acc, B, ATM, $d>)
69      ]
70
71  rule bank_receive_transaction:
72      [
73          !Account(Acc, B, pkApp, pin),
74          In(<Acc, B, ATM, d>),
75          Fr(~nonce_app)
76      ]
77      --[NonceChosenByBank(~nonce_app)]->
78      [
79          BankWaitForConfirmation(Acc, B, ATM, d, ~
                nonce_app),
```

```
80          Out(aenc(<Acc, B, ATM, d, ~nonce_app>, pkApp))
81      ]
82
83  rule app_receive_transaction:
84      [
85          !Account(Acc, B, pkApp, pin),
86          !App_Ltk(pkApp, k),
87          In(c)
88      ]
89      --[]->
90      [
91          AppDecryptedTransactionData(Acc, B, adec(c, k))
92      ]
93
94  rule user_verify_transaction_app:
95      [
96          AppDecryptedTransactionData(Acc, B, <Acc, B, ATM,
                d, nonce>),
97          UserWaitForConfirmation(Acc, B, ATM, d)
98      ]
99      --[UserLeakedNonce(nonce)]->
100     [
101         Out(nonce),
102         UserWaitForConfirmationATM(Acc, B, ATM, d)
103     ]
104
105 rule atm_user_enter_pin_and_nonce:
106     [
107         !Account(Acc, B, pkApp, pin),
108         UserWaitForConfirmationATM(Acc, B, ATM, d),
109         ATMTransactionStarted(Acc, B, ATM, d),
110         In(nonce_app)
111     ]
112     --[ ]->
113     [
114         Out_C(ATM, B, <Acc, B, ATM, d, nonce_app, pin>)
115     ]
116
117 rule atm_attacker_enter_pin_and_nonce:
118     [
119         !Account(Acc, B, pkApp, pin),
120         AttackerWaitForConfirmation(Acc, B, ATM, d),
121         ATMTransactionStarted(Acc, B, ATM, d),
122         In(nonce_app),
123         In(pin)
124     ]
125     --[ AttackerConfirmedTransaction(Acc, B, ATM, d) ]->
126     [
127         Out_C(ATM, B, <Acc, B, ATM, d, nonce_app, pin>)
```

```
128     ]
129
130 rule compromised_atm_leak_pin:
131     [
132         !ATM_Compromised(ATM),
133         !Account(Acc, B, pkApp, pin),
134         UserWaitForConfirmationATM(Acc, B, ATM, d),
135         ATMTransactionStarted(Acc, B, ATM, d)
136     ]
137     --[ WithdrawAtCompromisedATM(Acc, B), PINLeaked(pin)
            ]->
138     [
139         Out(pin)
140     ]
141
142 rule bank_verify_transaction:
143     [
144         BankWaitForConfirmation(Acc, B, ATM, d, nonce_app
                ),
145         In_C(ATM, B, <Acc, B, ATM, d, nonce_app, pin>),
146         !Account(Acc, B, pkApp, pin)
147     ]
148     --[BankVerifiedTransaction(Acc, B, ATM, d)]->
149     [
150     ]
151
152 rule compromise_atm:
153     [
154         !ATM(ATM)
155     ]
156     --[ CompromiseATM(ATM) ]->
157     [
158         !ATM_Compromised(ATM)
159     ]
160
161 rule compromise_app:
162         [
163             !Account(Acc, B, pk_app, pin),
164             !App_Ltk(pk_app, k_app)
165         ]
166         --[ CompromiseApp(Acc, B) ]->
167         [
168             Out(k_app)
169         ]
170
171 lemma types [sources]:
172     "(All nonce #i. UserLeakedNonce(nonce) @i
173     ==>
174     (
```

```
175         (Ex #j. NonceChosenByBank(nonce) @j) |
176         (Ex #j. KU(nonce) @j & j < i)
177     ))"
178
179 /* Confidential channel rules */
180 rule ChanOut_C:
181     [
182         Out_C($A,$B,x)
183     ]
184     --[ ChanOut_C($A,$B,x) ]->
185     [
186         !Conf($B,x)
187     ]
188
189 rule ChanIn_C:
190     [
191         !Conf($B,x),
192         In($A)
193     ]
194     --[ ChanIn_C($A,$B,x) ]->
195     [
196         In_C($A,$B,x)
197     ]
198
199 rule ChanIn_CAdv:
200     [
201         In(<$A,$B,x>)
202     ]
203     -->
204     [
205         In_C($A,$B,x)
206     ]
207
208 // These restrictions are not required for security but
        simplify proofs and counterexamples
209 restriction UniqueAccounts:
210         "All acc bank #i #j. AccountRegistered(acc, bank)
                @i & AccountRegistered(acc, bank) @j ==> #i
              = #j"
211
212 restriction UniqueATMs:
213          "All atm #i #j. ATMCreated(atm) @i & ATMCreated(
               atm) @j ==> #i = #j"
214
215 #ifdef SANITY
216 lemma end_of_line_reached_without_attack:
217     exists-trace
218     "Ex acc bank atm data #i. BankVerifiedTransaction(acc
        , bank, atm, data) @i & not (Ex #j. CompromiseApp
```

```
        (acc, bank) @j) & not (Ex #k. CompromiseATM(atm)
        @k) & not (Ex #l. MaliciousTransactionInitiated(
        acc, bank, atm, data) @#l)"
lemma multiple_banks:
    exists-trace
        "Ex bank1 bank2 #t1 #t2. BankCreated(bank1) @t1 &
            BankCreated(bank2) @t2 & (not bank1 = bank2)
           "
lemma multiple_atms:
    exists-trace
        "Ex ATM1 ATM2 #t1 #t2. ATMCreated(ATM1) @ #t1 &
           ATMCreated(ATM2) @ #t2 & not ATM1 = ATM2"

lemma user_can_register_with_multiple_banks:
    exists-trace
        "Ex Acc Bank1 Bank2 #t1 #t2. AccountRegistered(
           Acc, Bank1) @ #t1 & AccountRegistered(Acc,
           Bank2) @ #t2 & not Bank1 = Bank2"

lemma atm_leaks_pin:
    exists-trace
        "Ex pin #i. PINLeaked(pin) @#i"

lemma attacker_can_start_transaction:
    exists-trace
        "Ex acc bank atm data #i.
           MaliciousTransactionInitiated(acc, bank, atm,
            data) @#i"

lemma attacker_initiated_transaction_accepted:
    exists-trace
        "Ex acc bank atm data #i #j.
           MaliciousTransactionInitiated(acc, bank, atm,
            data) @#i & BankVerifiedTransaction(acc,
           bank, atm, data) @#j & not (Ex #k.
           TransactionInitiated(acc, bank, atm, data) @#
           k)"

lemma compromised_atm_can_confirm_transaction:
    exists-trace
        "Ex acc bank atm data #i. BankVerifiedTransaction
           (acc, bank, atm, data) @#i & not (Ex #j.
           TransactionInitiated(acc, bank, atm, data) @#
           j) & not (Ex #k.
           MaliciousTransactionInitiated(acc, bank, atm,
            data) @#k)"

lemma compromise_both_devices:
    exists-trace
```

```
248         "Ex acc bank atm data #i. BankVerifiedTransaction
               (acc, bank, atm, data) @#i & not (Ex #j.
               TransactionInitiated(acc, bank, atm, data) @#
               j)"
249
250 #endif
251
252 // one-out-of-two security
253 #ifdef SEC
254 lemma only_user_initiated_transactions_accepted:
255     "All acc bank data atm #i. BankVerifiedTransaction(
           acc, bank, atm, data) @ #i ==> ((Ex #j. (
           TransactionInitiated(acc, bank, atm, data) @ #j))
256     | (Ex #j #k. CompromiseApp(acc, bank) @#j &
           WithdrawAtCompromisedATM(acc, bank) @#k))"
257
258 lemma replay_attack_impossible:
259     "All acc1 bank1 data1 atm1 acc2 bank2 data2 atm2 #i #
           j. BankVerifiedTransaction(acc1, bank1, atm1,
           data1) @i & BankVerifiedTransaction(acc2, bank2,
           atm2, data2) @j & not #i = #j
260 ==>
261 ( (Ex #k #l. TransactionInitiated(acc1, bank1, atm1,
       data1) @k & TransactionInitiated(acc2, bank2, atm2,
       data2) @l & not #k = #l)
262 | (Ex #m #n. CompromiseApp(acc1, bank1) @m &
       WithdrawAtCompromisedATM(acc1, bank1) @n)
263 | (Ex #m #n. CompromiseApp(acc2, bank2) @m &
       WithdrawAtCompromisedATM(acc2, bank2) @n)
264 )"
265 #endif
266
267 end
```

Appendix B

Tamarin Model for FIDO2D

Our Tamarin model for our protocol FIDO2D from Chapter 4 including the formalization of our security notion *one-out-of-two security* can be found below. For brevity, we omit sanity lemmas, as well as the variations of the model for users that do not compare transaction data as described in Section 4.5.4. However, the full Tamarin model is available online[1].

```
theory fido2d
begin
builtins: signing

rule new_server:
    [   ]
    --[ HonestServer($S) ]->
    [ !Honest($S) ]

rule register_first_device:
    [ Fr(~privkey)  ]
    --[ ]->
    [
        !Ltk_Dev1($I, $S, ~privkey),
        !Pk_Dev1($I, $S, pk(~privkey)),
        RegisteredPartially($I, $S),
        Out(<$I, pk(~privkey)>)
    ]

rule register_second_device:
    [
        Fr(~privkey),
        RegisteredPartially(I, S)
    ]
    --[ AccountRegistered(I, S) ]->
    [
        !Ltk_Dev2(I, S, ~privkey),
```

[1] https://tinyurl.com/2022-fido2d-tamarin

```
28          !Pk_Dev2(I, S, pk(~privkey)),
29          !Registered(I, S),
30          Out(<I, pk(~privkey)>)
31      ]
32
33
34  rule init_transaction:
35      [
36          !Registered(I, S),
37          !Honest(S)
38      ]
39      --[ TransactionBegin(I, S, $d) ]->
40      [
41          UserWaitForConfirmation(I, S, $d),
42          Dev1WaitForNonce(I, S, $d),
43          Out(<I, S, $d>)
44      ]
45
46  rule receive_transaction:
47      [
48          !Registered(I, S),
49          !Honest(S),
50          In(<I, S, d>),
51          Fr(~nonce)
52      ]
53      --[ TransactionReceived(I, S, d) ]->
54      [
55          ServerWaitForSignature(I, S, d, ~nonce),
56          Out_A(S, I, <I, d, ~nonce>)
57      ]
58
59  rule phish_transaction:
60      [
61          !Registered(I, S),
62          In(P)
63      ]
64      --[ TransactionBegin(I, S, $d), Phisher(P) ]->
65      [
66          UserWaitForConfirmation(I, S, $d),
67          Dev1WaitForNonce(I, P, $d),
68          Phished(I, S, P)
69      ]
70
71  rule receive_transaction_phisher:
72      [
73          In(<d, nonce>),
74          Phished(I, S, P)
75      ]
76      --[ ]->
```

```
77      [
78          Out_A(P, I, <I, d, nonce>)
79      ]
80
81  rule sign_nonce:
82      [
83          Dev1WaitForNonce(I, S, d),
84          In_A(S, I, <I, d, nonce>),
85          !Ltk_Dev1(I, S, privkey)
86      ]
87      --[ NonceSigned(I, S, nonce) ]->
88      [
89          Out(sign(<S, nonce>, privkey))
90      ]
91
92  rule verify_signature:
93      [
94          In(signature),
95          ServerWaitForSignature(I, S, d, nonce),
96          !Pk_Dev1(I, S, pubkey),
97          Fr(~nonce2)
98      ]
99      --[ Eq(verify(signature, <S, nonce>, pubkey), true),
            SignatureVerifiedDev1(I, S, d) ]->
100     [
101         ServerWaitForSecondSignature(I, S, d, ~nonce2),
102         Out_A(S, I, <S, I, d, ~nonce2>)
103     ]
104
105 rule sign_second_nonce:
106     [
107         UserWaitForConfirmation(I, S, d),
108         In_A(S, I, <S, I, d, nonce>),
109         !Ltk_Dev2(I, S, privkey)
110     ]
111     --[ DisplayData(I, S, d), NonceSigned(I, S, nonce) ]-
            >
112     [
113         Out(sign(<S, d, nonce>, privkey))
114     ]
115
116 rule verify_second_signature:
117     [
118         In(signature),
119         ServerWaitForSecondSignature(I, S, d, nonce),
120         !Pk_Dev2(I, S, pubkey)
121     ]
122     --[ Eq(verify(signature, <S, d, nonce>, pubkey), true
            ), TransactionComplete(I, S, d) ]->
```

```
123     [
124     ]
125
126 rule compromise_first_device:
127     [
128         !Ltk_Dev1(I, S, privkey)
129     ]
130     --[ CompromiseDev1(I, S) ]->
131     [
132         Out(privkey)
133     ]
134
135 rule compromise_second_device:
136     [
137         !Ltk_Dev2(I, S, privkey)
138     ]
139     --[ CompromiseDev2(I, S) ]->
140     [
141         Out(privkey)
142     ]
143
144 // Authentic Channel Rules from Tamarin Manual
145 rule ChanOut_A:
146     [
147         Out_A(A, B, x)
148     ]
149     --[ ChanOut_A(A, B, x) ]->
150     [
151         !Auth(A, x),
152         Out(<A, B, x>)
153     ]
154
155 rule ChanIn_A:
156     [
157         !Auth(A, x),
158         In(B)
159     ]
160     --[ ChanIn_A(A, B, x) ]->
161     [
162         In_A(A, B, x)
163     ]
164
165 restriction Equality:
166     "All x y #i. Eq(x,y) @i ==> x = y"
167
168 restriction HonestServersDontPhish:
169     "All server #i #j. HonestServer(server) @i & Phisher(
        server) @j ==> F"
170
```

```
171 // This restriction is not required for security but
        simplifies proofs and counterexamples
172 restriction UniqueAccounts:
173     "All acc server #i #j. AccountRegistered(acc, server)
            @i & AccountRegistered(acc, server) @j ==> #i =
           #j"
174
175 lemma only_user_initiated_transactions_accepted:
176     "All initiator transaction server #i.
           TransactionComplete(initiator, server,
           transaction) @i ==> ((Ex #j.
177            TransactionBegin(initiator, server, transaction)
               @j) | (Ex #k #l. CompromiseDev1(initiator,
               server) @k & CompromiseDev2(initiator, server
               ) @l))"
178
179 lemma replay_attack_impossible:
180     "All initiator1 transaction1 initiator2 transaction2
           server #i #j. TransactionComplete(initiator1,
           server, transaction1) @i & TransactionComplete(
           initiator2, server, transaction2) @j & not #i = #
           j ==> ((Ex #k #l. TransactionBegin(initiator1,
           server, transaction1) @k & TransactionBegin(
           initiator2, server, transaction2) @l & not #k = #
           l) | (Ex #m #n. CompromiseDev1(initiator1, server
           ) @m & CompromiseDev2(initiator1, server) @n) | (
           Ex #m #n. CompromiseDev1(initiator2, server) @m &
            CompromiseDev2(initiator2, server) @n) )"
181
182 end
```

Bibliography

[1] Dirk Achenbach, Roland Gröll, Timon Hackenjos, Alexander Koch, Bernhard Löwe, Jeremias Mechler, Jörn Müller-Quade, and Jochen Rill. "Your money or your life—modeling and analyzing the security of electronic payment in the UC framework". In: *International Conference on Financial Cryptography and Data Security*. 2019, pp. 243–261. DOI: 10.1007/978-3-030-32101-7_16 (cit. on pp. 39, 48, 84, 105).

[2] Wolfgang Ahrendt et al. "The KeY tool". In: *Software & Systems Modeling* 4.1 (2005), pp. 32–54. DOI: 10.1007/s10270-004-0058-x (cit. on p. 97).

[3] Adi Akavia, Dan Feldman, and Hayim Shaul. "Secure Search on Encrypted Data via Multi-Ring Sketch". In: *Proceedings of the 25th ACM Conference on Computer and Communications Security*. CCS. 2018 (cit. on p. 87).

[4] FIDO Alliance. *User Authentication Specifications Overview*. https://fidoalliance.org/specifications/. 2022. (Visited on 01/15/2023) (cit. on p. 12).

[5] Amnesty International. *When Best Practice Isn't Good Enough: Large Campaigns of Phishing Attacks in Middle East and North Africa Target Privacy-Conscious Users*. 2018. URL: https://www.amnesty.org/en/latest/research/2018/12/when-best-practice-is-not-good-enough/ (visited on 07/20/2020) (cit. on pp. 58, 60).

[6] Will Arthur, David Challener, and Kenneth Goldman. *A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security*. Springer Nature, 2015 (cit. on p. 99).

[7] Jason Aten. *Google Says 66% of Americans Still Do This 1 Thing That Puts Their Personal Information at a Huge Risk*. 2019. URL: https://www.inc.com/jason-aten/google-says-66-of-americans-still-do-this-1-thing-that-puts-their-personal-information-at-a-huge-risk-heres-how-google-wants-to-help.html (visited on 09/04/2020) (cit. on p. 57).

[8] Manuel Barbosa et al. “Provable security analysis of FIDO2”. In: *Annual International Cryptology Conference*. 2021, pp. 125–156 (cit. on pp. 67, 105).

[9] Elaine B. Barker. *Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms*. Tech. rep. NIST Special Publication (SP) 800-175B, Rev.1. Gaithersburg, MD: National Institute of Standards and Technology, 2020. DOI: 10.6028/NIST.SP.800-175Br1 (cit. on p. 9).

[10] Elaine B. Barker. *Recommendation for Key Management, Part 1: General*. Tech. rep. NIST Special Publication (SP) 800-57, Rev.5. Gaithersburg, MD, United States: National Institute of Standards & Technology, 2016. DOI: 10.6028/NIST.SP.800-57pt1r5 (cit. on p. 37).

[11] Mike Barnett et al. “Boogie: A modular reusable verifier for object-oriented programs”. In: *International Symposium on Formal Methods for Components and Objects*. 2005, pp. 364–387 (cit. on p. 97).

[12] David Basin, Saa Radomirovic, and Lara Schmid. “Modeling human errors in security protocols”. In: *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. 2016, pp. 325–340 (cit. on p. 79).

[13] David Basin, Ralf Sasse, and Jorge Toro-Pozo. “The EMV standard: Break, fix, verify”. In: *2021 IEEE Symposium on Security and Privacy*. 2021, pp. 1766–1781 (cit. on pp. 40, 56, 110).

[14] David A. Basin, Sasa Radomirovic, and Michael Schläpfer. “A Complete Characterization of Secure Human-Server Communication”. In: *IEEE 28th Computer Security Foundations Symposium, CSF 2015*. 2015, pp. 199–213. DOI: 10.1109/CSF.2015.21 (cit. on p. 54).

[15] Johes Bater et al. “SMCQL: Secure Query Processing for Private Data Networks.” In: *Proceedings of the VLDB Endowment* 10.6 (2017), pp. 673–684 (cit. on pp. 87, 105).

[16] Ingmar Baumgart, Matthias Borsig, Niklas Goerke, Timon Hackenjos, Jochen Rill, and Marek Wehmer. “Who controls your energy? on the (in) security of residential battery energy storage systems”. In: *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. 2019, pp. 1–6. DOI: 10.1109/SmartGridComm.2019.8909749 (cit. on p. 2).

[17] Lennart Beringer et al. “Verified Correctness and Security of OpenSSL HMAC”. In: *24th USENIX Security Symposium (USENIX Security 15)*. 2015, pp. 207–221 (cit. on p. 2).

[18] Karthikeyan Bhargavan et al. “Implementing TLS with Verified Cryptographic Security”. In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 445–459. DOI: 10.1109/SP.2013.37 (cit. on p. 2).

[19] Bruno Blanchet. *Introduction to cryptographic protocols*. `https://bblanche.gitlabpages.inria.fr/talks/VTSA11intro.pdf`. 2011. (Visited on 01/14/2023) (cit. on p. 12).

[20] Mike Bond et al. "Chip and Skim: Cloning EMV Cards with the Pre-play Attack". In: *2014 IEEE Symposium on Security and Privacy*. 2014, pp. 49–64. DOI: `10.1109/SP.2014.11` (cit. on pp. 4, 40, 42, 50, 52, 110).

[21] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. "Evaluating 2-DNF formulas on ciphertexts". In: *Theory of Cryptography Conference*. TCC. 2005 (cit. on pp. 11, 23, 36).

[22] Joseph Bonneau et al. "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes". In: *2012 IEEE Symposium on Security and Privacy*. 2012, pp. 553–567 (cit. on p. 82).

[23] Pietro Borrello et al. "ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture". In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 3917–3934 (cit. on pp. 87, 101).

[24] Jean-Ian Boutin. "The evolution of webinjects". In: *Virus Bulletin Conference*. 2014, pp. 25–34 (cit. on pp. 8, 9).

[25] Jo Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution". In: *27th USENIX Security Symposium (USENIX Security 18)*. USENIX. 2018 (cit. on pp. 87, 101).

[26] Michele Campobasso and Luca Allodi. "Impersonation-as-a-service: Characterizing the emerging criminal infrastructure for user impersonation at scale". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 1665–1680 (cit. on p. 60).

[27] R. Canetti. "Universally composable security: a new paradigm for cryptographic protocols". In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. 2001, pp. 136–145. DOI: `10.1109/SFCS.2001.959888` (cit. on p. 15).

[28] Ran Canetti and Marc Fischlin. "Universally composable commitments". In: *Annual International Cryptology Conference*. 2001, pp. 19–40 (cit. on p. 15).

[29] David Cash et al. "Highly-scalable searchable symmetric encryption with support for boolean queries". In: *Advances in Cryptology*. Crypto. 2013 (cit. on p. 23).

[30] Miguel Castro, Barbara Liskov, et al. "Practical byzantine fault tolerance". In: *OsDI*. Vol. 99. 1999. 1999, pp. 173–186 (cit. on p. 103).

[31] Luca Caviglione et al. “Tight arms race: overview of current malware threats and trends in their detection”. In: *IEEE Access* 9 (2020), pp. 5371–5396 (cit. on p. 98).

[32] David Cerdeira et al. “Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems”. In: *2020 IEEE Symposium on Security and Privacy*. 2020, pp. 1416–1432 (cit. on pp. 84, 101).

[33] Alberto Ceselli et al. “Modeling and assessing inference exposure in encrypted databases”. In: *ACM Transactions on Information and System Security (TISSEC)* 8.1 (2005), pp. 119–152 (cit. on p. 90).

[34] David Chaum. “Blind signatures for untraceable payments”. In: *Advances in cryptology*. 1983, pp. 199–203 (cit. on p. 55).

[35] David Chaum, Amos Fiat, and Moni Naor. “Untraceable Electronic Cash”. In: *CRYPTO ’88*. Vol. 403. LNCS. 1988, pp. 319–327. DOI: 10.1007/0-387-34799-2_25 (cit. on p. 55).

[36] Joey Chen. *WIP19 Espionage | New Chinese APT Targets IT Service Providers and Telcos With Signed Malware*. https://www.sentinelone.com/labs/wip19-espionage-new-chinese-apt-targets-it-service-providers-and-telcos-with-signed-malware/. 2022. (Visited on 01/14/2023) (cit. on p. 87).

[37] Tom Chothia et al. “Relay Cost Bounding for Contactless EMV Payments”. In: *Financial Cryptography and Data Security, FC 2015*. Vol. 8975. LNCS. 2015, pp. 189–206. DOI: 10.1007/978-3-662-47854-7_11 (cit. on p. 55).

[38] Yang-Wai Chow et al. “Authentication and transaction verification using QR codes with a mobile device”. In: *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. 2016, pp. 437–451 (cit. on pp. 83, 84, 105).

[39] Catalin Cimpanu. *Chinese hacker group caught bypassing 2FA*. 2019. URL: https://www.zdnet.com/article/chinese-hacker-group-caught-bypassing-2fa/ (visited on 07/20/2020) (cit. on p. 58).

[40] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. “On post-compromise security”. In: *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. 2016, pp. 164–178 (cit. on pp. 103, 104).

[41] Katriel Cohn-Gordon et al. “A formal security analysis of the signal messaging protocol”. In: *Journal of Cryptology* 33.4 (2020), pp. 1914–1983 (cit. on pp. 104, 105).

[42] Commonwealth Bank of Australia. *Cardless Cash*. 2018. URL: https://www.commbank.com.au/digital-banking/cardless-cash.html (visited on 09/25/2018) (cit. on p. 44).

[43] Véronique Cortier et al. “Designing and Proving an EMV-Compliant Payment Protocol for Mobile Devices”. In: *2017 IEEE European Symposium on Security and Privacy*. 2017, pp. 467–480. DOI: 10.1109/EuroSP.2017.19 (cit. on p. 56).

[44] Victor Costan and Srinivas Devadas. “Intel SGX Explained.” In: *Cryptology ePrint Archive* (2016) (cit. on p. 87).

[45] Cas Cremers et al. “A comprehensive symbolic analysis of TLS 1.3”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1773–1788 (cit. on pp. 15, 103).

[46] Cas Cremers et al. “Clone detection in secure messaging: improving post-compromise security in practice”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 1481–1495 (cit. on p. 104).

[47] Reza Curtmola et al. “Searchable symmetric encryption: improved definitions and efficient constructions”. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS. 2006 (cit. on pp. 11, 14, 21, 34, 35).

[48] Ivan Damgård et al. “Multiparty computation from somewhat homomorphic encryption”. In: *Annual Cryptology Conference*. 2012, pp. 643–662 (cit. on p. 102).

[49] Anupam Das et al. “The tangled web of password reuse.” In: *NDSS*. Vol. 14. 2014. 2014, pp. 23–26 (cit. on p. 57).

[50] Jean Paul Degabriele et al. “On the Joint Security of Encryption and Signature in EMV”. In: *CT-RSA 2012*. Vol. 7178. LNCS. 2012, pp. 116–135. DOI: 10.1007/978-3-642-27954-6_8 (cit. on p. 56).

[51] Michael Denzel, Alessandro Bruni, and Mark Dermot Ryan. “SmartGuard: Defending User Input from Malware”. In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. 2016, pp. 502–509. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0089 (cit. on p. 55).

[52] Giorgio Di Tizio, Michele Armellini, and Fabio Massacci. “Software Updates Strategies: a Quantitative Evaluation against Advanced Persistent Threats”. In: *IEEE Transactions on Software Engineering* (2022) (cit. on p. 98).

[53] Alexandra Dmitrienko et al. “On the (in) security of mobile two-factor authentication”. In: *International Conference on Financial Cryptography and Data Security*. 2014, pp. 365–383 (cit. on p. 102).

[54] Danny Dolev and Andrew Yao. "On the security of public key protocols". In: *IEEE Transactions on information theory* 29.2 (1983), pp. 198–208 (cit. on pp. 2, 15, 75, 109).

[55] Benjamin Dowling and Britta Hale. "There can be no compromise: The necessity of ratcheted authentication in secure messaging". In: *Cryptology ePrint Archive* (2020) (cit. on pp. 104, 105).

[56] Saar Drimer, Steven J Murdoch, and Ross Anderson. "Optimised to fail: Card readers for online banking". In: *International Conference on Financial Cryptography and Data Security*. 2009, pp. 184–200 (cit. on pp. ii, iii, 83, 84, 101, 105, 109).

[57] Saar Drimer and Steven J. Murdoch. "Keep Your Enemies Close: Distance Bounding Against Smartcard Relay Attacks". In: *16th USENIX Security Symposium (USENIX Security 07)*. 2007 (cit. on pp. 4, 40, 50, 51, 110).

[58] Duo. *One-Tap Authentication With Duo Push*. 2021. URL: `https://duo.com/product/multi-factor-authentication-mfa/authentication-methods/duo-push` (visited on 09/21/2021) (cit. on p. 83).

[59] Claudia Eckert. *IT-Sicherheit : Konzepte - Verfahren - Protokolle*. 10. Auflage. De Gruyter Studium. De Gruyter Oldenbourg, 2018. DOI: `10.1515/9783110563900` (cit. on p. 7).

[60] Martin Emms et al. "Harvesting High Value Foreign Currency Transactions from EMV Contactless Credit Cards Without the PIN". In: *2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014, pp. 716–726. DOI: `10.1145/2660267.2660312` (cit. on p. 55).

[61] EMV. *Integrated Circuit Card Specifications for Payment Systems: Book 1. Application Independent ICC to Terminal Interface Requirements, Version 4.3*. 2011 (cit. on pp. 40, 55).

[62] EMV. *Integrated Circuit Card Specifications for Payment Systems: Book 2. Security and Key Management, Version 4.3*. 2011 (cit. on pp. 40, 55).

[63] EMV. *Integrated Circuit Card Specifications for Payment Systems: Book 3. Application Specification, Version 4.3*. 2011 (cit. on pp. 40, 55).

[64] ENISA. *Flash note: EU cyber security agency ENISA; "High Roller" online bank robberies reveal security gaps*. 2012. URL: `https://www.enisa.europa.eu/news/enisa-news/copy_of_eu-cyber-security-agency-enisa-201chigh-roller201d-online-bank-robberies-reveal-security-gaps` (visited on 07/10/2021) (cit. on p. 2).

[65] Mark Ermolov and Maxim Goryachy. “How to Hack a Turned-Off Computer, or Running Unsigned Code in Intel Management Engine”. In: *Black Hat Europe*. 2017 (cit. on p. 87).

[66] European Commission. *Commission Delegated Regulation (EU) 2018/389*. 2017. URL: https://eur-lex.europa.eu/eli/reg_del/2018/389/oj (cit. on pp. 59, 83).

[67] Sky Faber et al. “Rich queries on encrypted data: Beyond exact matches”. In: *European Symposium on Research in Computer Security*. 2015, pp. 123–145 (cit. on p. 21).

[68] Adrienne Porter Felt et al. “A survey of mobile malware in the wild”. In: *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. 2011, pp. 3–14 (cit. on pp. 3, 9, 59, 60, 62, 99).

[69] FIDO Alliance. *Client to Authenticator Protocol (CTAP)*. 2021. URL: https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-20210615.pdf (cit. on pp. 13, 72).

[70] Unified Extensible Firmware Interface (UEFI) Forum. *Unified Extensible Firmware Interface (UEFI) Specification*. 2021. URL: https://uefi.org/sites/default/files/resources/UEFI_Spec_2_9_2021_03_18.pdf (cit. on p. 98).

[71] Aurelien Francillon et al. “Systematic treatment of remote attestation”. In: *Cryptology ePrint Archive* (2012) (cit. on p. 102).

[72] David Mandell Freeman. “Converting Pairing-Based Cryptosystems from Composite-Order Groups to Prime-Order Groups”. In: *Advances in Cryptology – EUROCRYPT 2010*. 2010, pp. 44–61 (cit. on p. 37).

[73] Nick Frymann et al. “Asynchronous Remote Key Generation: An Analysis of Yubico’s Proposal for W3C WebAuthn”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 939–954 (cit. on p. 73).

[74] Benjamin Fuller et al. “SoK: Cryptographically Protected Database Search”. In: *arXiv preprint 1703.02014* (2017) (cit. on p. 88).

[75] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. “The Bitcoin Backbone Protocol: Analysis and Applications”. In: *EUROCRYPT 2015*. 2015, pp. 281–310. DOI: 10.1007/978-3-662-46803-6_10 (cit. on p. 55).

[76] Filiz Sarah Gärtner. *Mobile Banking Via Display-TAN: Secure and Convenient*. 2015. URL: https://blog.gft.com/blog/2015/09/02/mobile-banking-via-display-tan-secure-and-convenient/ (visited on 01/15/2023) (cit. on p. 55).

[77] Tingjian Ge and Stan Zdonik. "Answering aggregation queries in a secure system model". In: *Proceedings of the 33rd international conference on Very Large Data Bases*. VLDB. 2007 (cit. on p. 21).

[78] Craig Gentry. "Fully Homomorphic Encryption Using Ideal Lattices". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. 2009, pp. 169–178. DOI: 10.1145/1536414.1536440 (cit. on pp. 10, 87).

[79] Peder Gjerstad et al. "Do President Trump's tweets affect financial markets?" In: *Decision Support Systems* 147 (2021), p. 113577. DOI: 10.1016/j.dss.2021.113577 (cit. on p. 81).

[80] Google. *Google's strongest security helps keep your private information safe*. 2021. URL: https://landing.google.com/advancedprotection/ (visited on 09/21/2021) (cit. on p. 83).

[81] Johannes Götzfried et al. "Cache attacks on Intel SGX". In: *Proceedings of the 10th European Workshop on Systems Security*. 2017, pp. 1–6 (cit. on pp. 87, 101).

[82] Paul A Grassi et al. *Digital Identity Guidelines (Authentication and Lifecycle Management)*. Tech. rep. NIST Special Publication (SP) 800-63b. Gaithersburg, MD: National Institute of Standards and Technology, 2017. DOI: 10.6028/NIST.SP.800-63b (cit. on p. 83).

[83] Chris Grier et al. "Manufacturing compromise: the emergence of exploit-as-a-service". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 821–832 (cit. on pp. 60, 97).

[84] Alex Grinman. *The Future of Krypton Is Here and It's Called Akamai MFA*. 2021. URL: https://krypt.co/blog/announcements/krypton-is-now-akamai-mfa.html (visited on 09/21/2021) (cit. on p. 83).

[85] Trusted Computing Group. *TCG PC Client Platform Firmware Profile Specification*. 2021. URL: https://trustedcomputinggroup.org/wp-content/uploads/TCG_PCClient_PFP_r1p05_v23_pub.pdf (cit. on p. 102).

[86] Paul Grubbs et al. "Leakage-Abuse Attacks against Order-Revealing Encryption". In: *Cryptology ePrint Archive* (2016) (cit. on p. 88).

[87] Christoph G. Günther. "An identity-based key-exchange protocol". In: *Advances in Cryptology — EUROCRYPT '89*. 1989, pp. 29–37. DOI: 10.1007/3-540-46885-4_5 (cit. on p. 103).

[88] Hakan Hacigümüş et al. "Executing SQL over encrypted data in the database-service-provider model". In: *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*. 2002. DOI: 10.1145/564691.564717 (cit. on pp. 88, 94, 95).

[89] Timon Hackenjos. "Secure Aggregation and Grouping in Encrypted Databases". MA thesis. Karlsruhe Institute of Technology (KIT), 2017 (cit. on p. 18).

[90] Timon Hackenjos, Florian Hahn, and Florian Kerschbaum. "SAGMA: Secure aggregation grouped by multiple attributes". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 587–601. DOI: 10.1145/3318464.3380569 (cit. on pp. 11, 18, 19, 87).

[91] Timon Hackenjos, Benedikt Wagner, Julian Herr, Jochen Rill, Marek Wehmer, Niklas Goerke, and Ingmar Baumgart. "FIDO2 With Two Displays—Or How to Protect Security-Critical Web Transactions Against Malware Attacks". In: *arXiv preprint arXiv:2206.13358* (2022). DOI: 10.48550/arXiv.2206.13358 (cit. on p. 57).

[92] Florian Hahn, Nicolas Loza, and Florian Kerschbaum. "Practical and Secure Substring Search". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018. DOI: 10.1145/3183713.3183754 (cit. on p. 21).

[93] Vincent Haupert and Stephan Gabert. "Short Paper: How to Attack PSD2 Internet Banking". In: *International Conference on Financial Cryptography and Data Security*. 2019, pp. 234–242 (cit. on pp. ii, iii, 66, 109).

[94] Vincent Haupert and Stephan Gabert. "Where to Look for What You See Is What You Sign? User Confusion in Transaction Security". In: *European Symposium on Research in Computer Security*. 2019, pp. 429–449 (cit. on p. 79).

[95] Vincent Haupert and Tilo Müller. "On App-based Matrix Code Authentication in Online Banking." In: *ICISSP*. 2018, pp. 149–160 (cit. on pp. 8, 58, 59, 62, 63, 83, 85).

[96] Chris Hawblitzel et al. "Ironclad Apps: End-to-End Security via Automated Full-System Verification". In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 2014, pp. 165–181 (cit. on pp. 2, 97).

[97] Jonas Hielscher et al. ""Taking out the Trash": Why Security Behavior Change requires Intentional Forgetting". In: *New Security Paradigms Workshop*. 2021, pp. 108–122 (cit. on p. 98).

[98] Muks Hirani, Sarah Jones, and Ben Read. *Global DNS Hijacking Campaign: DNS Record Manipulation at Scale*. 2019. URL: https://www.mandiant.com/resources/global-dns-hijacking-campaign-dns-record-manipulation-at-scale (visited on 11/22/2021) (cit. on p. 82).

[99] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. "A privacy-preserving index for range queries". In: *Proceedings of the 30th international conference on Very Large Data Bases*. VLDB. 2004 (cit. on p. 94).

[100] Bijit Hore et al. "Secure multidimensional range queries over outsourced data". In: *The VLDB Journal—The International Journal on Very Large Data Bases* 3 (2012), pp. 333–358 (cit. on p. 94).

[101] Yin Hu, William J. Martin, and Berk Sunar. "Enhanced flexibility for homomorphic encryption schemes via CRT". In: *Applied Cryptography and Network Security (ACNS)*. 2012 (cit. on p. 36).

[102] Troy Hunt. *Password reuse, credential stuffing and another billion records in Have I been pwned*. `https://www.troyhunt.com/password-reuse-credential-stuffing-and-another-1-billion-records-in-have-i-been-pwned/`. 2017. (Visited on 08/10/2020) (cit. on p. 60).

[103] Scott Ikeda. *Half a Million Zoom Accounts Compromised by Credential Stuffing, Sold on Dark Web*. 2020. URL: `https://www.cpomagazine.com/cyber-security/half-a-million-zoom-accounts-compromised-by-credential-stuffing-sold-on-dark-web/` (visited on 08/23/2020) (cit. on p. 60).

[104] Abdullah Imran et al. "SARA: Secure Android Remote Authorization". In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 1561–1578 (cit. on p. 100).

[105] Iulia Ion, Rob Reeder, and Sunny Consolvo. ""... no one can hack my mind": Comparing Expert and Non-Expert Security Practices". In: *Eleventh Symposium On Usable Privacy and Security (SOUPS)*. 2015, pp. 327–346 (cit. on p. 58).

[106] Charlie Jacomme and Steve Kremer. "An extensive formal analysis of multi-factor authentication protocols". In: *ACM Transactions on Privacy and Security (TOPS)* 24.2 (2021), pp. 1–34 (cit. on pp. 58, 82, 83).

[107] Luke Jenkins et al. *Suspected Russian Activity Targeting Government and Business Entities Around the Globe*. 2021. URL: `https://www.mandiant.com/resources/blog/russian-targeting-gov-business` (visited on 11/06/2022) (cit. on p. 79).

[108] Arnold Johnson et al. *Guide for Security-Focused Configuration Management of Information Systems*. Tech. rep. NIST Special Publication (SP) 800-128. Gaithersburg, MD: National Institute of Standards and Technology, 2011. DOI: `10.6028/NIST.SP.800-128` (cit. on p. 8).

[109] Seny Kamara and Tarik Moataz. “SQL on structurally-encrypted databases”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT. 2018 (cit. on pp. 19, 21, 88, 94).

[110] George Karantzas and Constantinos Patsakis. “An empirical assessment of endpoint detection and response systems against advanced persistent threats attack vectors”. In: *Journal of Cybersecurity and Privacy* 1.3 (2021), pp. 387–421 (cit. on p. 98).

[111] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Second Edition. CRC press, 2014. DOI: 10.5555/2700550 (cit. on pp. 1, 2, 10, 13, 103).

[112] Eric Klieme et al. “FIDOnuous: A FIDO2/WebAuthn Extension to Support Continuous Web Authentication”. In: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2020, pp. 1857–1867 (cit. on p. 72).

[113] Alexander Koch. “Cryptographic protocols from physical assumptions”. PhD thesis. Karlsruhe Institute of Technology (KIT), 2019 (cit. on p. 39).

[114] Radhesh Krishnan Konoth, Victor van der Veen, and Herbert Bos. “How anywhere computing just killed your phone-based two-factor authentication”. In: *International Conference on Financial Cryptography and Data Security*. 2016, pp. 405–421 (cit. on pp. 64, 102).

[115] Radhesh Krishnan Konoth et al. “SecurePay: Strengthening Two-Factor Authentication for Arbitrary Transactions”. In: *2020 IEEE European Symposium on Security and Privacy*. 2020, pp. 569–586 (cit. on pp. 52, 58, 62, 73, 77, 84, 100, 105).

[116] Kat Krol et al. “"They brought in the horrible key ring thing!" Analysing the Usability of Two-Factor Authentication in UK Online Banking”. In: *USEC Workshop on Usable Security at the Network and Distributed System Security Symposium*. 2015 (cit. on pp. 59, 83).

[117] Ralf Küsters, Tomasz Truderung, and Jürgen Graf. “A Framework for the Cryptographic Verification of Java-Like Programs”. In: *2012 IEEE 25th Computer Security Foundations Symposium*. 2012, pp. 198–212. DOI: 10.1109/CSF.2012.9 (cit. on p. 2).

[118] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine generals problem”. In: *Concurrency: the works of leslie lamport*. 2019, pp. 203–226 (cit. on p. 103).

[119] Yehuda Lindell. “How to simulate it–a tutorial on the simulation proof technique”. In: *Tutorials on the Foundations of Cryptography* (2017), pp. 277–346 (cit. on p. 14).

[120] Sanam Ghorbani Lyastani et al. “Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication”. In: *2020 IEEE Symposium on Security and Privacy*. 2020, pp. 268–285 (cit. on p. 59).

[121] m0chan. *Bypassing 2FA For Fun With Evilginx2*. 2019. URL: https://m0chan.github.io/2019/07/26/Bypassing-2FA-For-Fun-With-Evilginx2.html (visited on 08/05/2020) (cit. on p. 60).

[122] Valentin JM Manès et al. “The art, science, and engineering of fuzzing: A survey”. In: *IEEE Transactions on Software Engineering* 47.11 (2019), pp. 2312–2331 (cit. on p. 97).

[123] Mohammad Mannan and Paul C Van Oorschot. “Leveraging personal devices for stronger password authentication from untrusted computers”. In: *Journal of Computer Security* 19.4 (2011), pp. 703–750 (cit. on pp. 83, 84, 101, 105).

[124] René Mayrhofer et al. “The android platform security model”. In: *ACM Transactions on Privacy and Security (TOPS)* 24.3 (2021), pp. 1–35 (cit. on pp. 71, 98).

[125] Simon Meier et al. “The TAMARIN Prover for the Symbolic Analysis of Security Protocols”. In: *Computer Aided Verification*. 2013, pp. 696–701 (cit. on pp. 5, 48, 56, 73, 84).

[126] Robert Merget et al. “Scalable scanning and automatic classification of TLS padding oracle vulnerabilities”. In: *28th USENIX Security Symposium (USENIX Security 19)*. 2019, pp. 1029–1046 (cit. on p. 15).

[127] Microsoft. *How User Account Control works*. 2022. URL: https://learn.microsoft.com/en-us/windows/security/identity-protection/user-account-control/how-user-account-control-works (visited on 12/10/2022) (cit. on p. 8).

[128] Nick Mooney, Nick Steele, and Jeremy Erickson. *Network Transport Summary*. 2020. URL: https://github.com/w3c/webauthn/files/4588654/The.Network.Transport.May.2020.pdf (visited on 08/29/2020) (cit. on p. 73).

[129] Steven J. Murdoch et al. “Chip and PIN is Broken”. In: *31st IEEE Symposium on Security and Privacy*. 2010, pp. 433–446. DOI: 10.1109/SP.2010.33 (cit. on pp. ii, iii, 40, 50, 109, 110).

[130] Muhammad Naveed, Seny Kamara, and Charles V. Wright. “Inference attacks on property-preserving encrypted databases”. In: *Proceedings of the 22nd ACM Conference on Computer and Communications Security*. CCS. 2015 (cit. on pp. ii, iii, 19, 20, 88, 94, 109, 111).

[131] Terry Nelms et al. “Towards measuring and mitigating social engineering software download attacks”. In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016, pp. 773–789 (cit. on p. 60).

[132] *Trial of John Buckley, Thomas Shenton*. 1781. URL: https://www.oldbaileyonline.org/browse.jsp?div=t17810912-37 (visited on 09/22/2018) (cit. on p. 39).

[133] Michele Orru and Giuseppe Trotta. *Muraena: The Unexpected Phish*. 2019. URL: https://conference.hitb.org/hitbsecconf2019ams/materials/D2T1%20-%20Muraena%20-%20The%20Unexpected%20Phish%20-%20Michele%20Orru%20&%20Giuseppe%20Trotta.pdf (visited on 07/20/2020) (cit. on p. 60).

[134] Pascal Paillier. "Public-key cryptosystems based on composite degree residuosity classes". In: *Advances in Cryptology — EUROCRYPT '99*. 1999, pp. 223–238 (cit. on pp. 10, 21).

[135] Antonis Papadimitriou et al. "Big Data Analytics over Encrypted Datasets with Seabed." In: *12th USENIX Symposium on Operating Systems Design and Implementation*. OSDI. 2016 (cit. on pp. 20, 88, 89, 92, 94, 95, 105).

[136] Vasilis Pappas, Michalis Polychronakis, and Angelos D Keromytis. "Transparent ROP exploit mitigation using indirect branch tracing". In: *22nd USENIX Security Symposium (USENIX Security 13)*. 2013, pp. 447–462 (cit. on p. 97).

[137] Torben Pryds Pedersen. "Non-interactive and information-theoretic secure verifiable secret sharing". In: *Annual international cryptology conference*. 1991, pp. 129–140 (cit. on p. 102).

[138] Raluca Ada Popa et al. "CryptDB: protecting confidentiality with encrypted query processing". In: *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*. SOSP. 2011 (cit. on pp. 20, 88, 94, 95, 105, 111).

[139] Pengfei Qui et al. "VoltJockey: Abusing the Processor Voltage to Break Arm TrustZone". In: *GetMobile: Mobile Computing and Communications* 24.2 (2020), pp. 30–33 (cit. on pp. 84, 101).

[140] Michael O. Rabin. "How to Exchange Secrets with Oblivious Transfer". In: *Cryptology ePrint Archive* (1981) (cit. on p. 102).

[141] Raiffeisenbank Böllingertal eG. *VR-mobileCash: Geld abheben ohne Karte*. URL: https://www.raiba-boellingertal.de/privatkunden/girokonto-kreditkarten/service/geld-abheben-ohne-karte.html (visited on 01/16/2023) (cit. on pp. 44, 105).

[142] Anil Rao. *Rising to the Challenge—Data Security with Intel Confidential Computing*. 2022. URL: https://community.intel.com/t5/Blogs/Products-and-Solutions/Security/Rising-to-the-Challenge-Data-Security-with-Intel-Confidential/post/1353141 (visited on 01/03/2023) (cit. on p. 101).

[143] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. RFC Editor, 2018. URL: https://www.rfc-editor.org/rfc/rfc8446.txt (cit. on p. 12).

[144] Jochen Rill. "Towards Applying Cryptographic Security Models to Real-World Systems". PhD thesis. Karlsruhe Institute of Technology (KIT), 2020 (cit. on p. 39).

[145] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126 (cit. on p. 9).

[146] RSA. *Making Sense of Man-in-the-browser Attacks: Threat Analysis and Mitigation for Financial Institutions*. 2010. URL: https://viewer.media.bitpipe.com/1039183786_34/1295277188_16/MITB_WP_0510-RSA.pdf (visited on 07/22/2020) (cit. on pp. 58, 61).

[147] Keegan Ryan. "Hardware-backed heist: extracting ECDSA keys from Qualcomm's TrustZone". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 181–194 (cit. on pp. 84, 101).

[148] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted execution environment: what it is, and what it is not". In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. 2015, pp. 57–64 (cit. on p. 100).

[149] Benedikt Schmidt et al. "Automated analysis of Diffie-Hellman protocols and advanced security properties". In: *2012 IEEE 25th Computer Security Foundations Symposium*. 2012, pp. 78–94 (cit. on pp. 15, 16).

[150] Bruce Schneier. "Two-factor authentication: too little, too late". In: *Communications of the ACM* 48.4 (2005), p. 136 (cit. on p. 58).

[151] Smart Card Alliance. *Contactless EMV Payments: Benefits for Consumers, Merchants and Issuers*. URL: https://www.emv-connection.com/downloads/2016/06/Contactless-2-0-WP-FINAL-June-2016.pdf (visited on 12/17/2018) (cit. on p. 43).

[152] National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*. Tech. rep. Federal Information Processing Standards Publications (FIPS PUBS) 197. Gaithersburg, MD, 2001. DOI: 10.6028/NIST.FIPS.197 (cit. on p. 9).

[153] Maddie Stone and Clement Lecigne. *How we protect users from 0-day attacks*. 2021. URL: https://blog.google/threat-analysis-group/how-we-protect-users-0-day-attacks/ (visited on 09/21/2021) (cit. on pp. 8, 60).

[154] The Tamarin Team. *Tamarin-Prover Manual*. 2021. URL: https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf (cit. on pp. 15–18, 76).

[155] Kurt Thomas et al. “Data breaches, phishing, or malware? Understanding the risks of stolen credentials”. In: *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2017, pp. 1421–1434 (cit. on pp. 8, 59, 62).

[156] Mathy Vanhoef and Frank Piessens. “Key reinstallation attacks: Forcing nonce reuse in WPA2”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1313–1328 (cit. on p. 1).

[157] Verizon. *Data Breach Investigations Report 2022*. 2022. URL: `https://www.verizon.com/business/resources/reports/2022/dbir/2022-data-breach-investigations-report-dbir.pdf` (cit. on pp. 2, 8, 97).

[158] Visa. *Visa Token Service*. URL: `https://usa.visa.com/partner-with-us/payment-technology/visa-token-service.html` (visited on 12/17/2018) (cit. on p. 43).

[159] Melanie Volkamer, Martina Angela Sasse, and Franziska Boehm. “Analysing simulated phishing campaigns for staff”. In: *European Symposium on Research in Computer Security*. 2020, pp. 312–328 (cit. on p. 98).

[160] Daniel Votipka et al. “Hackers vs. testers: A comparison of software vulnerability discovery processes”. In: *2018 IEEE Symposium on Security and Privacy*. 2018, pp. 374–391 (cit. on p. 97).

[161] Paul Georg Wagner, Pascal Birnstill, and Jürgen Beyerer. “Establishing secure communication channels using remote attestation with TPM 2.0”. In: *International Workshop on Security and Trust Management*. 2020, pp. 73–89 (cit. on p. 102).

[162] Stephan Wiefling, Markus Dürmuth, and Luigi Lo Iacono. “What’s in Score for Website Users: A Data-driven Long-term Study on Risk-based Authentication Characteristics”. In: *International Conference on Financial Cryptography and Data Security*. 2021, pp. 361–381 (cit. on p. 60).

[163] Stephan Wiefling, Luigi Lo Iacono, and Markus Dürmuth. “Is this really you? An empirical study on risk-based authentication applied in the wild”. In: *IFIP International Conference on ICT Systems Security and Privacy Protection*. 2019, pp. 134–148 (cit. on p. 62).

[164] World Wide Web Consortium. *Web Authentication: An API for accessing Public Key Credentials Level 2*. 2021. URL: `https://www.w3.org/TR/webauthn-2/` (cit. on pp. 13, 73, 83, 102, 105).

[165] Andrew C Yao. “Protocols for secure computations”. In: *23rd annual symposium on foundations of computer science (sfcs 1982)*. 1982, pp. 160–164 (cit. on p. 102).

[166] Fengwei Zhang and Hongwei Zhang. “SoK: A study of using hardware-assisted isolated execution environments for security”. In: *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. 2016, pp. 1–8 (cit. on p. 100).

[167] Wenting Zheng et al. “Opaque: An Oblivious and Encrypted Distributed Analytics Platform.” In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 283–298 (cit. on p. 87).

Author's Publications

[1] Dirk Achenbach, Roland Gröll, Timon Hackenjos, Alexander Koch, Bernhard Löwe, Jeremias Mechler, Jörn Müller-Quade, and Jochen Rill. "Your money or your life—modeling and analyzing the security of electronic payment in the UC framework". In: *International Conference on Financial Cryptography and Data Security*. 2019, pp. 243–261. DOI: 10.1007/978-3-030-32101-7_16 (cit. on pp. 39, 48, 84, 105).

[16] Ingmar Baumgart, Matthias Borsig, Niklas Goerke, Timon Hackenjos, Jochen Rill, and Marek Wehmer. "Who controls your energy? on the (in) security of residential battery energy storage systems". In: *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. 2019, pp. 1–6. DOI: 10.1109/SmartGridComm.2019.8909749 (cit. on p. 2).

[90] Timon Hackenjos, Florian Hahn, and Florian Kerschbaum. "SAGMA: Secure aggregation grouped by multiple attributes". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 587–601. DOI: 10.1145/3318464.3380569 (cit. on pp. 11, 18, 19, 87).

[91] Timon Hackenjos, Benedikt Wagner, Julian Herr, Jochen Rill, Marek Wehmer, Niklas Goerke, and Ingmar Baumgart. "FIDO2 With Two Displays—Or How to Protect Security-Critical Web Transactions Against Malware Attacks". In: *arXiv preprint arXiv:2206.13358* (2022). DOI: 10.48550/arXiv.2206.13358 (cit. on p. 57).