Attack Once, Compromise All? On the Scalability of Attacks

Eva Hetzel[1(✉)], Marc Nemes[2], and Jörn Müller-Quade[1]

[1] KASTEL Security Research Labs, Karlsruhe Institute of Technology, Karlsruhe, Germany
{eva.hetzel,joern.mueller-quade}@kit.edu
[2] FZI Research Center for Information Technology, Karlsruhe, Germany
nemes@fzi.de

Abstract. Electronic voting schemes are often criticized for being insecure, on the grounds that a successful attack would allow an adversary to manipulate all votes at once. It is argued that attacks therefore have a higher impact at lower adversary costs compared to paper-based schemes, where attacks are cumbersome. In this paper, we propose a framework to quantify how prone different protocols are to attacks that scale well. For this purpose, we introduce the notion of *scalability of attacks*. We give the adversary access to an oracle which can break common cryptographic building blocks and assumptions and analyze how many inputs of a (multiparty computation) protocol they can learn or manipulate for each oracle access. The more inputs are affected, the more susceptible the protocol is to attacks that scale well. We compare several pairs of protocols solving the same problem in different ways in three examples and analyze the scalability of attacks on each protocol. We find that some protocols have a fatal breakdown, i.e. all inputs are affected with only one access to the oracle, while other protocols scale linearly or have a threshold, where the number of affected inputs increases drastically from one access to the other. Our framework provides strong arguments in favoring one voting scheme over another. It enables voting authorities to compare schemes that appear equally secure at first glance, and to consider the scalability of attacks when deciding on a scheme.

Keywords: Scalability · Risk Assessment · Quantification of Security · E-Voting

1 Introduction

Electronic voting schemes are often criticized for being insecure. One point that is mentioned frequently is that if an adversary can hack a voting server, they can manipulate all votes at once. Therefore, an attack on an electronic voting scheme has better scalability than an attack on a paper-based voting scheme. Others argue that electronic voting schemes provide cryptographic proofs of correctness

D. Duenas-Cid et al. (Eds.): E-Vote-ID 2025, LNCS 16028, pp. 90–106, 2026.
https://doi.org/10.1007/978-3-032-05036-6_6

which are virtually impossible to forge, nullifying this scalability point of the discussion.

The security of electronic voting schemes and other cryptographic protocols is always based on assumptions. These assumptions are based on presumably (unproven) difficult mathematical problems (e.g., integer factorization) or trust assumptions (e.g., the majority of all participants is honest) among others. If a single assumption is no longer met once, cryptographers generally consider the entire protocol to be insecure. Two different protocols with unmet assumptions are then considered as being equally insecure, as in, both protocols are 100% insecure. In this paper, we analyze what happens if some assumptions are no longer met only in a few instances, e.g., when a state-sponsored adversary can find a collision in a weak cryptographic hash function at great expense and within a month's time (using a lot of conventional computers or a quantum computer). Our goal is to quantify whether the adversary can manipulate the outcome of an entire election or only manipulate the vote of a single voter. When comparing this scenario to another cryptographic protocol, where one can easily find collisions for all used hashes, we hypothesize that the former protocol is more resilient against widespread attacks than the latter.

1.1 Our Contribution

Our contributions are the following:

- We introduce the notion of *scalability of attacks*. Intuitively, the scalability describes how the adversary's cost and the damage of the attack evolve with multiple executions.
- We give a formal definition of scalability using oracles that can be used by the adversary to break assumptions.
- We compare the scalability of attacks on the confidentiality and integrity of several exemplary cryptographic protocols, e.g., voting schemes.

1.2 Related Work

In the following, an overview of related work regarding the notion of scalability of attacks and related cryptographic topics is given.

In 2010, Herley [4] introduced a distinction between scalable and non-scalable attacks. He considers an attack to be scalable, if the adversary's cost grows more slowly than linearly in the number of affected instances, or in his case, internet users. However, he considers the personalization of scalable attacks, e.g., through the personalization of phishing emails, not to be profitable, which might no longer hold true in times of artificial intelligence. Furthermore, we consider the term scalability to have more nuances than linear vs. faster growth of the adversary's cost, some of which we present in Table 1 and Sect. 4.

Another approach by Lopuhaä-Zwakenberg and Stoelinga [6] discussing the relation of the adversary's cost with the damage caused by an attack is using

attack trees. However, this approach lacks an analysis of how the cost evolves after multiple executions of an attack.

Next to these explicit studies of the scalability of attacks, there are related topics in cryptography. While in a classical black-box adversarial model, the adversary does not learn intermediate results or other local data of parties involved in a cryptographic protocol, *leakage-resilient cryptography* focuses on protocols that remain secure even if some local, secret data is communicated to the adversary by a side-channel or other vulnerabilities [5]. In a similar fashion, our work looks at the consequences of some secret data being known by the adversary.

Quantum annoying protocols are protocols that can be broken by quantum computers but require additional effort like solving discrete logarithms. So far, this property has only been defined for password-authenticated key exchange protocols (PAKEs) [3,10], while a broader definition applicable to other types of protocols is expected to be published soon. A PAKE is quantum-annoying, if a classical adversary has to solve one discrete logarithm for each password guess when breaking the protocol. To test the quantum annoying property of a protocol, the adversary has access to a discrete logarithm oracle, similar to the oracle we define for scalability in this paper. Both properties consider attacks in the context of broken assumptions and evaluate the security that is left. Therefore, results concerning one notion can be translated to the other: If a PAKE is quantum annoying, attacks on the protocol scale badly, since the oracle needs to be queried for every password.

The rest of the paper is structured as follows: Sect. 2 introduces preliminaries. The formal definition including oracles that break (cryptographic or other) assumptions is given in Sect. 3. Section 4 presents three exemplary protocol comparisons regarding the scalability of attacks on their confidentiality and integrity. Section 5 concludes the paper.

2 Preliminaries

Before a formal definition of attack scalability can be given, this section presents the necessary preliminaries and assumptions. The focus of this paper lies on the scalability of attacks on (cryptographic) protocols with multiple parties participating in the protocol execution. The goal of the adversary is to either learn some new information about the parties' inputs or to manipulate the output of the protocol without being detected. This corresponds to attacks on the confidentiality and integrity of the protocol, respectively. While we do not consider other security properties like availability in our analysis, the scalability of attacks on those properties can be considered in a similar manner.

The scalability of an attack quantifies how the adversary's cost and the damage of the attack relate to one another when the protocol is attacked multiple times. To determine the cost, we introduce an oracle. This oracle is capable of breaking various assumptions that the security of the protocol relies on. The

adversary is allowed to access this oracle before, during or after the execution of the protocol and the cost of the attack is measured in oracle accesses. Furthermore, the adversary may be one of the parties carrying out the protocol. In order to compare different protocols, we need to determine how many units, e.g., secret inputs are affected per oracle access for both protocols. This corresponds to the damage of an attack. This number of affected units per oracle access can be a constant value or scale differently, e.g., linearly, exponentially, in a logarithmic manner etc. We show example protocols with different attack scalabilities in Sect. 4.

Plotting the number of oracle accesses against the number of affected units results in a scaling curve. All scaling curves are weakly monotonically increasing, because additional oracle accesses can never decrease the information gained or damage caused by the adversary.

The scalability of an attack on a specific protocol does not have to be a continuous function. If, for example, the first k oracle accesses do not provide any usable information to the adversary, but with the information gained from a $(k+1)$th oracle access the adversary learns all inputs, then we get a discontinuous function which jumps from one constant to another. Alternatively, the curve could grow linearly for the first k oracle accesses and then begin to grow quadratically for each additional oracle access.

In this paper, we look at assumptions that might be wrong, e.g., based on unproven mathematical assumptions or human error, as well as assumptions based on unconditional or information theoretical security. While the latter would be impossible to break using a computationally bounded oracle, we extend the notion of using an oracle to actions like breaking in and stealing a secret. We also disregard other attacks that might be possible without an oracle while having a similar impact and therefore be the greater risk in the real world. Further, we do not estimate how much effort it would be to build or run such an oracle, we just assume the existence of one.

We do not compare the severity of the attack made possible by the oracle, other than the number of affected units. For example, we do not determine whether learning the vote of a single voter is better or worse than changing a single vote to a desired candidate or learning which candidate a specific voter certainly did not vote for. We justify this decision as follows: In the United States, about a third of the voters disclose openly which candidate they are voting for [7] but especially in swing states, every vote counts. Therefore, integrity seems to be an important security property for US elections. Meanwhile, Germany has a 5% electoral threshold for its federal election and no swing states, making the confidentiality of the ballot possibly more important than the integrity of a few votes which do not change the outcome of the election anyway. Which protocol is better with the same scalability of attacks therefore also depends on the voters. Accordingly, we analyze attacks on both the confidentiality and the integrity of protocols separately.

3 Formal Definition

In this section we formally define the scalability of attacks. Let $\mathcal{O}$ be an oracle that can break any computationally secure cryptographic building block, e.g., performing integer factorization, finding a discrete logarithm for any group, extracting a private key from a public key, finding a pre-image for a given hash or generating the required randomness to unveil a computationally binding commitment to any desired message. We also allow the oracle to break unconditionally or information-theoretically secure cryptographic building blocks like decrypting a one-time-pad or unveiling a perfectly hiding commitment scheme.

Let $I \in \mathbb{N}_0$ be the number of inputs or pieces of sensitive data, e.g., a voter's vote or inputs for a multiparty computation. Let $k \in \mathbb{N}_0$ be how many times the adversary accesses the oracle $\mathcal{O}$ at any point.

We introduce the notion of attack scalability of a security property of a protocol, i.e. how well attacks on the property scale when some assumptions are no longer fulfilled. Let

$$\begin{aligned} S : \mathbb{N}_0 &\to \mathbb{N}_0 \\ k &\mapsto (i)_{0 \leq i \leq I} \end{aligned}$$

be the scalability of attacks on a protocol using the oracle $\mathcal{O}$ k times.

We define attacking the confidentiality as learning at least one bit of new information about an input before (e.g., during the setup), during or after the execution of the protocol. We define attacking the integrity as changing at least one bit of an input without being detected until the execution of the protocol is finished or changing at least one bit of an input which cannot be recovered.

We use indices to denote the respective protocol and attack dimension, e.g., $S^{\text{Integrity}}_{\text{BingoVoting}}$ denotes the scalability of attacks on the integrity of the Bingo Voting protocol. We also define $S = \max(S^{\text{Integrity}}, S^{\text{Confidentiality}})$.

The higher S is, the better attacks scale. In this context, "good" scalability means good in the eyes of the adversary. If an attack scales well, the adversary only needs to put a small amount of effort into their attack to affect many inputs. A person choosing a protocol for implementation should therefore always aim for the scheme with the worse scalability. However, as the following section shows, which scheme has the worse attack scalability is not always obvious.

Table 1. Examples of the scalability of attacks depending on the number of oracle accesses k given an oracle $\mathcal{O}$

$S(1) = I$	Accessing $\mathcal{O}$ once, the adversary can affect (manipulate or learn) all inputs.
$S(k) = \lfloor \frac{k}{2} \rfloor$	For every other oracle access, the adversary can affect an input.
$S(k) = \begin{cases} 0 & k < 5 \\ I & \text{otherwise} \end{cases}$	Until accessing the oracle five times, the adversary cannot affect a single input. Once they reach the threshold of five accesses, they can affect all inputs.

Table 1 shows some example values for scalability. As mentioned in the previous section, $S(k) \geq S(k'), k > k'$ holds.

4 Examples

In this section we show three examples to demonstrate different scalabilities of attacks. In each example, we compare two protocols solving the same problem or very similar problems in different ways and show the impact of an oracle on the confidentiality and the integrity of the respective protocol. We demonstrate the best known (to us) attacks for each protocol. Future research may uncover new attacks on the protocols shown and not shown here and therefore change the scalability of (known) attacks, similar to the discovery of new attacks on (symmetric) encryption schemes.

4.1 Public Key Infrastructure Vs. Web of Trust

First, we compare the scalability of attacks on a certificate authority (CA) in a public key infrastructure (PKI) with a web of trust (WoT).

A certificate authority uses their private key in order to sign certificates which map a public key to a website, person or institution. The public key can belong to another CA which results in a tree of signed certificates, with a root CA at the top. People trusting the root CA can use the verified chain of signed certificates in order to trust a certificate signed by a lower CA. Due to the high number of people trusting the root CA, keeping its private key secret is an important task. If an adversary obtains the private key of a root CA (e.g., by accessing the oracle once), they can affect a large number of people.

In a web of trust, each participant signs the public keys of other participants they have met after personally verifying their identity. If a participant wants to contact a participant they have never encountered, they can check whether there is a transitive connection to the target starting with trusted and verified peers. The trustworthiness of each public key and other thresholds can be parametrized and used to limit the number of hops and therefore the number of public keys that are considered authentic.

In this example, we chose I to be the number of people that want to securely communicate with each other. More precisely, we look at the following workflow that the adversary aims to attack:

1. Alice wants to send a message m to Bob.
2. Alice visits a public key server and searches for the public key of Bob.
3. The public key server sends Alice Bob's public key pk_{Bob} as well as a certificate $C_{\text{Bob}} = (pk_{\text{Bob}}, \text{Bob}, \{\sigma_1, ..., \sigma_n\})$ which includes Bob's name and a list of signatures σ_i, $i = 1, ..., n$, which show that this public key belongs to Bob.
4. Alice verifies the signatures of C_{Bob} and aborts if there are too few valid signatures. Otherwise, Alice continues.

5. Alice generates a symmetric key K_s and encrypts the message m using K_s. She then encrypts the symmetric key K_s using Bob's public key pk_{Bob} and signs the message using her private key sk_{Alice}. Alice then sends the signed, encrypted message, the encrypted symmetric key, and a certificate $C_{\text{Alice}} = (pk_{\text{Alice}}, \text{Alice}, \{\sigma_1^*, ..., \sigma_m^*\})$ to Bob (*hybrid encryption*).
6. Bob checks the validity of C_{Alice} and the validity of the signature, then decrypts the symmetric key using his secret key sk_{Bob} and then decrypts the encrypted message using the symmetric key K_s.

In order to verify the receiver's public key before sending a message, the users could establish a PKI or a WoT. In the case of a PKI, the set of signatures σ_i in the certificates contains a single element (signed by the certificate authority). We allow the adversary to intercept messages that are sent in the network, because an adversary with access to an oracle could be considered powerful enough to control parts of the network.

PKI. To attack the confidentiality and integrity of Alice's message to Bob in a PKI, the adversary intercepts the message from the public key server to Alice and replaces Bob's public key with $pk_{\text{Adversary}}$.[1] Additionally, the adversary creates a new certificate and signs it (using the CA's secret key which they received from the oracle) which convincingly shows that $pk_{\text{Adversary}}$ belongs to Bob. Next, the adversary intercepts the message that Alice sends to Bob and decrypts it using $sk_{\text{Adversary}}$. The adversary can read the message m, manipulate it before encrypting and signing it using $sk_{\text{Adversary}}$ and create a new certificate $C'_{\text{Alice}} = (pk_{\text{Adversary}}, \text{Alice}, \sigma')$ that shows that $pk_{\text{Adversary}}$ belongs to Alice. After attaching the new certificate, they send the new message to Bob, who considers the message to be authentic. With all users trusting the certificates issued by the root CA we get $S_{\text{PKI}}(1) = S_{\text{PKI}}^{\text{Integrity}}(1) = S_{\text{PKI}}^{\text{Confidentiality}}(1) = I$.

WoT. If an adversary manages to get the private key of any participant of the web of trust (also by accessing the oracle once), they can affect all participants who strongly trust this participant and therefore indirectly some additional participants. The adversary can use this private key to issue certificates and make other participants trust in the authenticity of $pk_{\text{Adversary}}$, just as in the case of a PKI. However, it is unlikely that there is a single person in a web of trust that could affect a majority of the participants. Ulrich et al. [11] as well as Richters and Peixoto [8] analyzed the OpenPGP web of trust graph. According to them, authenticated communication in a WoT is only possible within strongly connected components. The largest strongly connected component in the OpenPGP WoT graph consisted of 39796 nodes, i.e. keys that have been signed or are used for signing other participants' keys, in 2009 [8]. The indegree of a node shows how many users confirm the authenticity of the respective key by signing it. The highest indegree of a single node in the largest strongly

[1] For simplicity's sake, we assume the communication with the public key server to be unencrypted.

connected component was 965 [8]. By learning the private key of this node, the adversary could directly affect 965 participants, assuming these participants also put high trust in signatures of this participant and ignoring indirectly affected participants in order to get a rough estimate.[2] The average indegree was 7.58 [8], which is due to a large majority of the nodes having a very low indegree of 1 or 2. Since the information about the network topology, including the best connected nodes is available at the keyservers, we assume that the adversary picks the nodes with the highest indegrees first. Therefore, with the first oracle access, the scalability results in $S_{\mathrm{WoT}}(1) = \frac{965}{39796} I \approx 0.024I$.

Figure 1 illustrates the portion of the participants affected by the first 100 oracle accesses. To get an upper bound on the scalability, we assume that the number of affected participants increases with every oracle access according to the indegree of the node chosen in this step, disregarding that some of the participants might already have been affected in previous attack steps. While an attack on the root CA in a PKI affects every certificate issued by it, the web of trust uses a decentralized approach which leads to an approximately logarithmic scalability behavior.

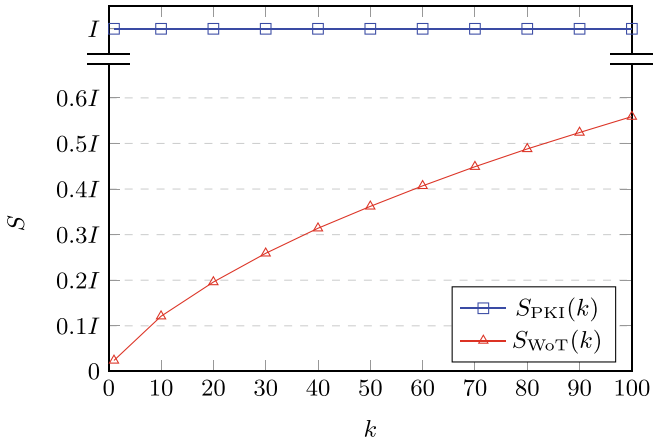

Fig. 1. Scalability of attacks on authentication infrastructures. The scaling curves for attacks on the confidentiality and integrity coincide for both the PKI and the WoT.

In this scenario, confidentiality and integrity are both compromised with the presented attack. By intercepting and decrypting messages encrypted with the adversary's public key $pk_{\mathrm{Adversary}}$, the adversary gets to read confidential messages. If they go even further by manipulating the message and attaching a forged signature, the integrity of the message is broken as well.

[2] We also treat each key as representing a different participant, which is a simplification since some participants register multiple keys. This affects the scalability by a small factor but does not undermine the argument.

Figure 1 considers the adversary not to be the CA or part of the WoT. However, it is not unlikely that the adversary participates in the WoT and that other participants have trust in their key. If this was the case, the curve should be shifted to the left by one oracle access, since the adversary knows their own private key from the beginning. Allowing the adversary to be the CA in a PKI, on the other hand, would instantly break the whole authentication infrastructure. Without accessing the oracle, the adversary would possess the CA's private key, making the scheme meaningless.

4.2 Mix-Net Voting Vs. Bingo Voting

We now compare two voting schemes: Mix-net Voting and Bingo Voting. We allow the adversary to be part of the voting authority, because an adversary with access to an oracle could be considered powerful enough to infiltrate the voting authority. We chose I to be the number of votes. While the integrity of some votes may be irrelevant in a winner-take-all system, the manipulation of a single vote might be non-negligible, for example, if a candidate receives additional funding based on the number of votes they received in their last election.

Mix-Net Voting. Homomorphic encryption combined with re-encryption mix-nets can be used for anonymous voting [2]. Several distrusting parties jointly generate a secret key / public key pair for an ElGamal encryption scheme, so that no party knows the entire secret key, and publish the public key. Voters encrypt their vote using this public key and post the encrypted ballot on a public bulletin board. The encrypted ballots are then sent through a mix-net consisting of several mix-servers run by also distrusting parties. The mix-servers shuffle and rerandomize the votes before publishing them. Lastly, the secret key is recombined using enough of the key shares and the votes are decrypted and counted. In order to prove the correctness, each mix-server also publishes a proof that it did not alter the contents of any ballot.

To manipulate the tally, any mix-server can alter the content of a ballot and then forge the proof of correctness by using the oracle to solve the Decisional Diffie-Hellman-Problem in the Chaum-Pedersen Protocol used for the proofs. Therefore, $S^{\text{Integrity}}_{\text{Mix-netVoting}}(k) = k$.

Since the public key is public, a single oracle access is enough to learn the complete secret key, even though the secret key is shared among several distrusting parties. The adversary can then decrypt all public votes on the bulletin board and learn the vote of all voters before the ballots are shuffled. Therefore, $S^{\text{Confidentiality}}_{\text{Mix-netVoting}}(1) = I$ and therefore $S_{\text{Mix-netVoting}}(1) = I$.

Bingo Voting. We briefly explain the Bingo Voting protocol by Bohli et al. [1]. Bingo Voting consists of three phases:

Phase 1: Pre-voting Phase. Let l denote the number of candidates. For every candidate P_i, n random numbers are generated, where n is the number of eligible voters. They are combined into tuples (N^i_j, P_i) and act as dummy

votes for the upcoming voting phase. Unconditionally hiding commitments to the tuples are created using fresh randomness for each commitment and published on a public bulletin board. Furthermore, the voting authority proves that the dummy votes are equally distributed among all candidates.

Phase 2: Voting Phase. The voter presses the respective button on the voting machine in order to vote for their preferred candidate. A trusted random number generator generates a new number (which is indistinguishable from the random numbers used for the dummy votes), displays it to the voter and transfers it to the voting machine, which assigns it to the chosen candidate. For each other candidate, the voting machine picks a dummy vote from the pool of dummy votes for this candidate. Then, the voting machine prints a receipt for the voter, which lists all candidates next to a random number. The voter verifies that the number next to their chosen candidate matches the number on the display of the random number generator.

Phase 3: Post-voting Phase. After counting all votes, the voting authority publishes the following information:

- The final result of the tally.
- All receipts printed by the voting machine during the voting phase.
- All unused commitments with their respective unveil-information. These are dummy votes that were not used because a voter voted for this candidate or because of voter abstention.
- Proofs of correctness, e.g., for each receipt, prove that it contains a proper dummy vote for each candidate except for the selected candidate.

In order to prove the correctness, the voting authority performs the following steps for each ballot:

1. A new tuple with the chosen candidate and the random value from the random number generator is created.
2. A commitment to this new tuple is published together with the $l - 1$ commitments to the dummy votes of the ballot is computed. Therefore, l commitments are published, one of them is a new commitment and all other commitments can be found in the list of all dummy votes. This list is called C_{left}.
3. C_{left} is rerandomized, shuffled and published as C_{middle}.
4. C_{middle} is rerandomized, shuffled and published as C_{right}.
5. All commitments in C_{right} are unveiled. Therefore, everyone can verify that the contents of the commitments are exactly the contents of the receipt, including the actual fresh random number (indistinguishable from the rest). The only difference is the order of the candidates, due to the shuffle.
6. Using an (external) coin toss, either the random value from rerandomizing and shuffling to get from C_{left} to C_{right} or from C_{middle} to C_{right} is revealed (the voting authority may not predict the coin toss). If the voting authority tries to cheat, they will be detected with a probability of 50% per manipulated proof. Since the association between C_{middle} and the third, unused list is not revealed, the association between C_{left} and C_{right} stays hidden.

If the adversary is part of the voting authority and uses the oracle to break the binding property of the commitments, they can manipulate the tally: Suppose they want candidate A to win the election. When a voter votes for a candidate other than A, instead of using a dummy ballot meant for A, they pick a dummy ballot meant for the chosen candidate. When unveiling, they use the oracle to generate randomness which unveils the commitment to the value they wish. Therefore, for every oracle access, they can manipulate a vote without being detected, resulting in $S^{Integrity}_{\text{BingoVoting}}(k) = k$.

If the adversary uses the oracle to unveil a commitment, they can compare the unveiled random number from the commitment with the random numbers on voters' ballots and potentially find out which candidate a voter did not vote for, which also leads to $S^{Confidentiality}_{\text{BingoVoting}}(k) = k$. Therefore, $S_{\text{BingoVoting}}(k) = k$.

Figure 2 illustrates the scalabilities of attacks on the two voting schemes. While attacks on the integrity of both schemes scale linearly, there is a big difference regarding attacks on the confidentiality. The secret key used to decrypt the votes in Mix-net Voting is a single point of failure of the entire scheme, if an adversary is powerful enough to compute it from the public key. Bingo Voting, on the other hand, does not have such a vulnerability and attacks on the confidentiality scale linearly as well. Attacks on the Mix-net Voting protocol therefore have a better scalability than attacks on the Bingo Voting protocol.

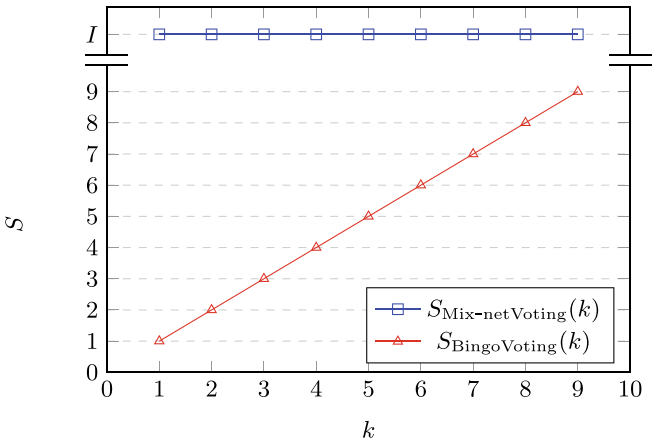

Fig. 2. Scalability of attacks on voting schemes. For Mix-net Voting, the scalability of the attack on the confidentiality is displayed, which is also the overall (maximum) scalability of the scheme. For Bingo Voting, the scalability of attacks on the confidentiality and integrity coincide.

4.3 Secret Sharing

In this section, different ways of storing a secret on one or multiple servers are compared. This secret can later be retrieved, e.g., to use it as a backup if the local

copy of the secret is lost. The secret can be a decryption key, for example. The data to be protected is the secret itself. Since we also consider partial leakage of the secret, we divide the secret into information units, which are the substrings that the secret comprises. For the following examples, the secret is assumed to be divided into five substrings and therefore, $I = 5$.

One Server. The trivial option for sharing a secret is to store it on a server unencrypted. In this case, $\mathcal{O}$ can be understood as an adversary, e.g., a secret agent, stealing the secret from the server. Here, $S^{\text{Confidentiality}}_{\text{OneServer}}(1) = I$, since the entire secret can be learned with only one oracle access.

The same holds, if the adversary wants to target the integrity of the secret instead of the confidentiality. With one oracle access, the secret can be changed at the server, therefore $S^{\text{Integrity}}_{\text{OneServer}}(1) = I$ and $S_{\text{OneServer}}(1) = I$. We assume that the manipulation of the secret is not detected until after the protocol execution.[3]

Substrings. Another option is to divide the secret into substrings and send them to several servers. The substrings correspond to the five information units that make up I. To guarantee being able to retrieve the secret even if some of the servers are unresponsive, each share is sent to two servers. Since the secret is divided into five substrings, where each is shared with two servers, this leads to ten servers used in total. Since the adversary does not know which servers store the same shares and which ones could give them new information, the adversary has to guess the shares they want to get from the oracle. $S^{\text{Confidentiality}}_{\text{Substrings}}(1) = 1$ for the first oracle access, since one substring, i.e., one information unit of the secret is learned, independent of the server that was attacked. $\mathcal{O}$ is the same oracle as in the single server case: It can be used by the adversary to steal the secret or a share of the secret from one server at a time. For the second oracle access, with a probability of $\frac{1}{9}$, nothing new is learned, because the server with the same share as the first one was chosen. With a probability of $\frac{8}{9}$, however, a new share containing one substring of the secret is discovered. This leads to $E[S^{\text{Confidentiality}}_{\text{Substrings}}(2)] = 1 + \frac{1}{9} \cdot 0 + \frac{8}{9} \cdot 1 = \frac{17}{9}$. Figure 3 shows the expected portion of the secret learned after every oracle access.

If the adversary's goal is not to learn the secret but to change the secret, the attack scales differently. If the integrity of the secret is targeted, the adversary can use the oracle to manipulate the shares stored at the servers. However, the adversary does not know which servers will be queried later to reconstruct the secret and which ones therefore have an impact on the integrity of the whole secret if their shares are manipulated. With every oracle access, one server can be attacked. This results in $E[S^{\text{Integrity}}_{\text{Substrings}}(k)] = 0.5 \cdot k$, since the manipulated

[3] At the end of the protocol execution, the manipulated secret should be indistinguishable from the real one. However, if e.g., the secret was a decryption key, the decryption with the manipulated key might either fail or lead to a different plaintext. In this case, mechanisms could be implemented to recover the real secret, which are out of the scope of this work.

share is only chosen with probability $\frac{1}{2}$ when reconstructing the secret and only one substring of the secret can be manipulated at a time. If one substring of the secret is manipulated, it would still be possible to recover the remaining four substrings of the secret by the end of the protocol.

Looking at the second oracle access, there are four possibilities: With the first oracle access, either a share was picked that will be used when reconstructing the secret or one that will not be used. If the share will be used, the second oracle access can either lead to an additional share that will also be used for reconstruction with probability $\frac{4}{9}$, or one that will not be used with probability $\frac{5}{9}$. The same happens if the first pick was a miss: With the next oracle access, the adversary can pick a share that will be selected for reconstruction and therefore have an impact on the integrity of the secret with probability $\frac{5}{9}$, or one that will not be selected with probability $\frac{4}{9}$. This leads to a scalability of $E[S^{\text{Integrity}}_{\text{Substrings}}(2)] = \frac{1}{2} + \frac{1}{2} \cdot \frac{4}{9} \cdot 1 + \frac{1}{2} \cdot \frac{5}{9} \cdot 1 = 1$ in the second oracle access. For the following oracle accesses, a similar behavior can be observed. The resulting linear curve is shown in Fig. 3.

Shamir's Secret Sharing. A more advanced approach is the well-known secret sharing scheme by Shamir [9]. The idea of Shamir's secret sharing is to send shares of the secret to n servers and only t, $t \leq n$ servers are necessary to reconstruct the secret. While there was a part of the secret leaked to every server in the previous approach, any $t-1$ servers working together cannot learn anything about the secret when using Shamir's secret sharing. With this approach, the shares do not correspond to information units anymore. Therefore, the adversary must steal the shares of t servers to learn the secret. However, in this approach, it does not matter which servers the adversary steals shares from, since the shares all differ from each other and any t shares are enough to reconstruct the secret. The scalability can therefore be described as $S^{\text{Confidentiality}}_{\text{Shamir'sSecretSharing}}(k) = \begin{cases} 0 & k < t \\ I & \text{otherwise} \end{cases}$.

To compare the two secret sharing approaches using multiple servers in Fig. 3, we choose $n = 10$ and $t = 5$. Values of t closer to n might however be more appropriate in practice.

Targeting the integrity of the secret instead of its confidentiality, the adversary already succeeds with probability $\frac{1}{2}$ at the first oracle access. This results from choosing five out of the ten servers when reconstructing the secret. The compromised share is chosen with probability $\binom{9}{4}/\binom{10}{5} = 0.5$ and this results in $E[S^{\text{Integrity}}_{\text{Shamir'sSecretSharing}}(1)] = 0.5 \cdot I$. With Shamir's secret sharing scheme, the entire secret is compromised if one of the shares is manipulated. At the second oracle access, the probability that at least one of the compromised shares is used to reconstruct the secret is $(\binom{10}{5} - \binom{8}{5})/\binom{10}{5}$. Therefore, $E[S^{\text{Integrity}}_{\text{Shamir'sSecretSharing}}(2)] \approx 0.78 \cdot I$. Figure 3 shows the expected value of the scalability of the attack on the integrity of Shamir's secret sharing scheme.

The solid lines in Fig. 3 show that while the adversary does not learn anything about the secret until the fifth oracle query when using Shamir's secret sharing,

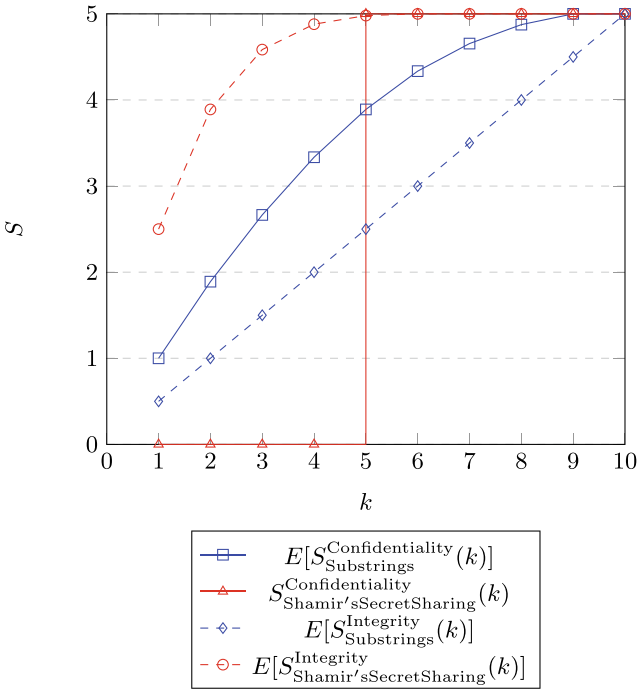

Fig. 3. Scalability of attacks on secret sharing schemes when using ten servers. Attacks on confidentiality as well as integrity are considered. For the substring approach and the integrity of Shamir's secret sharing scheme, expected values are displayed.

they are expected to learn between $\frac{3}{5}$ and $\frac{4}{5}$ of the secret with the same number of queries when substrings were sent to the servers instead. If the adversary is lucky and picks servers guarding different substrings with every oracle access, they might even be able to recover the entire secret within five queries. Depending on how valuable parts of the secret already are to the adversary, they might find the scalability of the substring approach favorable. The substrings might e.g., be meaningful secrets by themselves, or the secret might be a key and with every substring gained, brute-forcing becomes less complex. However, we assume that the final goal of the adversary is to learn the entire secret. Which scheme should be implemented therefore strongly depends on the application. In different scenarios, an adversary might prefer the scalability behavior of Shamir's secret sharing scheme, since they can be sure to know the whole secret after five oracle accesses. Here it should be noted, that if a higher value for t relative to the number of servers n is used, Shamir's secret sharing scheme provides stronger guarantees.

Comparing the scalability of the attacks on the integrity of the secret, the dashed lines in Fig. 3 show very different results. Since picking one compromised share when reconstructing the secret with Shamir's secret sharing scheme leads

to an entirely different secret, it scales much better from the first oracle access (in the adversary's view). Using the approach with the shared substrings, only parts of the secret can be manipulated with every access. Depending on the application, there might be a trade-off between the scalability of attacks on the confidentiality and the integrity of the schemes.

The plots in Fig. 3 refer to the case that the adversary is not the owner of any of the servers. If the adversary were to be one of the parties receiving shares, they would start with some information without having to access the oracle. Looking at the curves in Fig. 3, this would lead to them being shifted by one oracle access to the left.

5 Conclusion and Future Work

To sum up, we introduced the notion of scalability of attacks by giving a formal definition and comparing the scalability of attacks on the confidentiality and integrity of three different types of protocols. The notion of attack scalability can be used to quantify how the damage of an attack grows with multiple executions, also considering the adversary's cost. Since it is the adversary's goal to have a high impact with low costs, attacks that scale well in the adversary's view are more likely to occur. Investigating the scalability of attacks on a protocol can therefore reveal vulnerabilities and the notion presented in this paper is a valuable tool in risk assessment.

While our work confirmed that a single point of failure is a big disadvantage in protocols, the scalability of attacks is not a simple scalar value. When comparing two protocols, it might not even be directly clear which scalability is better. It depends on the application, the circumstances and the properties of the compared protocols. Voting authorities and other decision makers can use our work to compare contemplable protocols in their specific situation in order to make a more sophisticated decision.

In this paper, we focused on the integrity and confidentiality while giving the adversary a single oracle. For a more detailed comparison between two schemes, one could extend the notion of integrity to adding a ballot, removing a ballot, changing a ballot to a specific value or forcing the voter to vote for a random candidate. Confidentiality could also be further divided into learning a vote, learning which candidate a voter did certainly not vote for or which candidate they voted for with a specific probability. That way, one can better compare two schemes which both have linearly scaling attacks on their integrity at a first glance, but in reality, one is better protected against ballot stuffing. Denial-of-service attacks may be more difficult to formalize but could also be considered. Another dimension can be added by differentiating between oracles, e.g., one oracle that can only solve computational problems but cannot break information theoretically secure building blocks or another oracle that can attack supply chains (in order to replace a trusted random number generator with a predictable one).

Acknowledgments. This work was funded by the Topic Engineering Secure Systems of the Helmholtz Association (HGF) and supported by KASTEL Security Research Labs, Karlsruhe.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Bohli, J.M., Müller-Quade, J., Röhrich, S.: Bingo voting: secure and coercion-free voting using a trusted random number generator. In: E-Voting and Identity: First International Conference, VOTE-ID 2007, Bochum, Germany, October 4–5, 2007, Revised Selected Papers 1, pp. 111–124. Springer (2007)
2. Canetti, R., Hohenberger, S.: Special topics in cryptography, lecture 19: verifiable mix-net voting (2004)
3. Eaton, E., Stebila, D.: The quantum annoying property of password-authenticated key exchange protocols. In: Post-Quantum Cryptography: 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20–22, 2021, Proceedings 12, pp. 154–173. Springer (2021)
4. Herley, C.: The plight of the targeted attacker in a world of scale. In: WEIS (2010)
5. Kalai, Y.T., Reyzin, L.: A survey of leakage-resilient cryptography. In: Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, pp. 727–794. Association for Computing Machinery (2019)
6. Lopuhaä-Zwakenberg, M., Stoelinga, M.: Cost-damage analysis of attack trees. In: 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 545–558. IEEE (2023)
7. Maive, S.: Political campaign lawn sign survey - lawnstarter ranking (2024)
8. Richters, O., Peixoto, T.P.: Trust transitivity in social networks. PLoS ONE **6**(4), e18384 (2011)
9. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
10. Tiepelt, M., Eaton, E., Stebila, D.: Making an asymmetric PAKE quantum-annoying by hiding group elements. In: European Symposium on Research in Computer Security, pp. 168–188. Springer (2023)
11. Ulrich, A., Holz, R., Hauck, P., Carle, G.: Investigating the OpenPGP web of trust. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 489–507. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23822-2_27