REACTIVE: Rethinking Effective Approaches Concerning Trustees in Verifiable Elections

Josh Benaloh[1], Michael Naehrig[1], and Olivier Pereira[1,2]

[1] Microsoft Research, Redmond, WA, USA
[2] UCLouvain, B-1348 Louvain-la-Neuve, Belgium

Abstract. For more than forty years, two principal questions have been asked when designing verifiable election systems: how will the integrity of the results be demonstrated and how will the privacy of votes be preserved? Many approaches have been taken towards answering the first question such as use of MixNets and homomorphic tallying. But in the academic literature, the second question has always been answered in the same way: decryption capabilities are divided amongst multiple independent "trustees" so that a collusion is required to compromise privacy.
In practice, however, this approach can be fairly challenging to deploy. Human trustees rarely have a clear understanding of their responsibilities, and they typically all use identical software for their tasks. Rather than exercising independent judgment to maintain privacy, trustees are often reduced to automata who just push the buttons they are told to when they are told to, doing little towards protecting voter privacy.
This paper looks at several aspects of the trustee experience. It begins by discussing various cryptographic protocols that have been used for key generation in elections, explores their impact on the role of trustees, and notes that even the theory of proper use of trustees is more challenging than it might seem. This is illustrated by showing that one of the only references defining a full threshold distributed key generation (DKG) for elections defines an insecure protocol. Belenios claims to rely on that reference for its DKG and security proof. Fortunately, it does not inherit the same vulnerability. We offer a security proof for the Belenios DKG.
The paper then discusses various practical contexts, in terms of humans, software, and hardware, and their impact on the practical deployment of a trustee-based privacy model.

1 Introduction

The academic approach to preserving privacy in verifiable election protocols is to rely on independent *trustees* who are each responsible for production, maintenance, use, and ultimately destruction of their own cryptographic keys.

Decades of literature offer creative cryptographic protocols with excellent properties to achieve precisely this kind of separation. The key ingredient, threshold encryption is a well-established technique that provides robustness by allowing a pre-determined number of keyholders to perform the necessary decryptions to complete an election while preventing any set of insufficient size from gaining

any information at all. There are efficient means of threshold encryption and many implementations of the details.

To implement this approach in practice, the trustees follow specified protocols for distributed key generation before the election, and must validate that the key they generated is actually used in the election. When encrypted tallies need to be decrypted after the voting period, they must validate a set of ciphertexts to be decrypted and follow protocols for threshold decryption. And, during the whole process, the trustees need to make sure that they remain in control of their secret key material, in order to prevent any abuse.

Human trustees are aided by trustee hardware, computing devices that run software for carrying out the necessary cryptographic operations and securely communicate with the other trustee devices. These protocols are often facilitated by a central authority such as an election administrator that routes the trustees inputs and outputs.

Ideally, to guarantee independence, trustees should provide their own independent hardware device and their own independently written software: corrupted hardware and software might display everything that a human trustee expects to see while leaking every secret to interested parties.

There is, however, an enormous problem in practice. Real trustees rarely—if ever—have the tools and expertise to exhibit the independent role assumed by these protocols.

In practice, trustees are often chosen from local communities as representatives of the public. They do not have the tools or the expertise to fulfill their responsibilities. They instead merely do what they are told—pressing the buttons they are told to press on the devices provided to them. There is no real independence and no substantive protection from a curious election administrator who wants to view election data that is supposed to remain confidential.

In other cases, trustees are instead chosen for their expertise. But even in these cases, they are usually required to utilize software and hardware provided to them by a single source they may not trust. And selecting trustees only from amongst "elites" may create suspicion from the public.

Of course, the ideal would be for trustees to be well-trained representatives of the public who utilize devices and software of their own choosing. But, as far as we know, this is not the case in today's deployments, and a generalized adoption of such practices seems unlikely at any time in the foreseeable future.

While the integrity of the results can potentially be verified over and over by independent parties without access to privileged information and at any chosen time, privacy currently depends on a closed set of trustees being able to complete their tasks diligently. As a result, without this true independent trustee expertise and hardware and software independence, there is very limited basis for confidence in the privacy of the votes.

This paper starts by reviewing cryptographic protocols for trustees, focusing on distributed key generation (DKG) protocols used in real-world verifiable elections. We discuss the properties of these protocols and the practical demands that they place on the trustees. On our way, we explore technical difficulties

in some DKGs and propose a fresh security analysis of the DKG protocol of Belenios.

We then turn to a discussion of the potential benefits and trade-offs that the reliance on secure hardware can bring. In particular, attestation mechanisms can shift the trust that currently needs to be placed in humans, with limited assessment and verification options, to verifiable attestations produced by hardware that can be independently reviewed and challenged.

2 Key Generation Protocols

Our starting point is the most common set of cryptographic protocols that trustees are expected to run in existing verifiable voting systems.

We focus on distributed key generation protocols as a central ingredient of the kind of tasks that trustees are expected to perform. There is of course more in the role of trustees than running a DKG, but the cryptographic operations associated to decryption bring essentially the same requirements as those for key generation, and are described elsewhere, so we do not systematically elaborate on them here. We will also elaborate on post-election operations below.

2.1 Single key

In its simplest form, the key pair used to encrypt ballots can be generated by one single entity. This approach has been used for a long time in the Estonian Internet voting system, in which all ballots are decrypted using a single key held in an HSM [8]. This solution relies entirely on the security of the HSM device: if the key is somehow extracted from the device, or if someone manages to ask that device to decrypt non-anonymized encrypted votes, then ballots are not secret anymore. If the HSM device fails and (access to) the decryption key is lost, the election tally cannot be computed. This second concern can be mitigated more easily with secret key backups. However, the existence of such backups creates new opportunities for stealing the secret key.

From a technical point of view, and in the context of ElGamal encryption, which is now overwhelmingly adopted for encrypting ballots, the generation process of a single key is extremely simple. A group $\mathbb{G}$ of prime order q is chosen (typically as a public parameter of the election system), together with a generator g of that group. The key generation consists in selecting a secret key $s \leftarrow \mathbb{Z}_q$ uniformly at random, and publishing $K = g^s$ as the encryption public key.

2.2 One-round DKG

To limit the risks of having the single decryption key stolen and potentially used to break the secrecy of all the votes, various voting systems turn to a very simple extension of the above key generation mechanism: n trustees $T_1, \ldots, T_n$ independently run the single key generation and publish the resulting public

keys. These are then (publicly) combined into a single encryption public key. Here, all n secret keys are needed for decryption.

Concretely, each T_i selects a secret key $s_i \leftarrow \mathbb{Z}_q$ as before and publishes the corresponding $K_i = g^{s_i}$ together with a Schnorr proof of knowledge of s_i in order to prevent so-called rogue key attacks [16]. The encryption public key is computed as $K = \prod_{i=1}^{n} K_i$, with a corresponding secret key $s = \sum_{i=1}^{n} s_i$ that is never explicitly computed.

This approach is used in various systems, including Helios [1] (since version 2.0), Belenios [10], and the Swiss Internet voting system [19]. Its use in the context of elections was analyzed by Bernhard et al. [6].

This protocol offers two important benefits. First, it addresses the privacy risk when relying only on a single decryption key: now, in order to break privacy, all n private keys are needed. Second, it keeps most of the simplicity of the single-key protocol. In particular, key generation can still be implemented in a dozen lines of Python code, which can be reviewed easily by a knowledgable trustee, even during a key generation ceremony.

On the flip side, this protocol is even more fragile than the single-key protocol: losing any one of the n secret keys is sufficient to prevent tallying of the election. This can of course again be addressed with backups, which may be less sensitive than in the single-key case since, again, a copy of every secret key is needed to perform decryption operations. Concretely, various solutions have been used: trustees can be paired, and each pair of trustees generates a single secret key of which two copies are kept. In some cases, more elegant solutions can be proposed: for instance, in a setting with 3 trustees sitting around a table, each trustee can give a copy of its secret key to the trustee sitting to its left. This, in effect, offers a two-out-of-three threshold key generation process.

2.3 Threshold DKG

A more general approach to handle the risks of some trustees (or secret keys) being unavailable, is to rely on a threshold distributed key generation protocol, in which only k out of n trustees are needed to perform a decryption operation. The protocols that have been deployed for elections all follow a structure proposed by Pedersen [18]: the n trustees start by generating their key pair as in the one-round DKG protocol outlined above, then run a verifiable secret sharing (VSS) protocol to share their secret key s_i with the other trustees, using a threshold of k. The trustees can combine the shares they received into k-out-of-n shares of the secret key s that is the sum of the s_i. The encryption public key K is computed exacty as in the one-round DKG above.

Variations of this protocol are used in various election protocols and have been implemented in software packages, including Verificatum [24], which relies on Gennaro et al. [12]'s protocol, Belenios [10], which relies on Cortier et al. [9]'s protocol and ElectionGuard [4], whose DKG protocol is analyzed by Benaloh et al. [5].

The obvious benefit these protocols have, is their flexibility for chosing the level of robustness: the original protocol by Pedersen and the one by Gennaro

et al. make it possible to choose any $k < n/2$, while the two other protocols aim at supporting any choice of $k \leq n$.

However, they are less convenient due to the additional complexity of the VSS. It is also necessary to exchange encrypted shares between trustees: they are required to all be present at the same time, or be present multiple times in order to complete the multiple rounds of the protocol. Code complexity also increases considerably because multiple rounds of inter-trustee secret and authentic communication must be handled: even an expert may be uncomfortable to review the code of a full threshold DKG during a key generation ceremony (of course, the review may happen in advance and code hashes can be compared, but this again brings additional complexity). This additional complexity may also have an impact on the election verifier: a verifier for Verificatum and Belenios will need to compute Lagrange coefficients and adjust for the set of trustees present for decryption, which is much less straightforward than in the one-round DKG protocol. ElectionGuard, though, administratively combines all the decryption proofs so that the verification of a decryption operation is as simple as in the single-key case. The focus on simplifying the process of election verification is explicit there.

This extra complexity in the key generation also leads to a more complicated security analysis, which we now illustrate in the context of the Belenios DKG.

2.4 The Belenios DKG

Belenios offers two DKG protocols: one that is essentially the single-round protocol above, and a threshold protocol that is claimed to be "described in [9] and proved in [9,6]" [13, p. 2].

As mentioned before, the threshold DKG protocol follows the Pedersen design, which we outline here, given a set of trustees $T_1, \ldots, T_n$, a chosen threshold k, and assuming the availability of a public bulletin board that behaves as a broadcast channel.

1. Each trustee T_i chooses a random polynomial $P_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,k-1}x^{k-1}$ of degree $k-1$ with coefficients in $\mathbb{Z}_q$ and posts $K_{i,j} = g^{a_{i,j}}$ for $j \in [0, k-1]$ on a bulletin board.
2. Each trustee T_i sends $s_{i,j} = P_i(j)$ to every other trustee T_j on a secret and authentic channel.
3. Each trustee T_j verifies the shares it received by checking that $g^{s_{i,j}} = \prod_{\ell=0}^{k-1} (K_{i,\ell})^{j^\ell}$ for every value of i. If any check fails, a recovery phase starts, which we do not discuss here.
4. The public key is defined as $K = \prod_{i=1}^{n} K_{i,0}$ and each trustee T_i computes its decryption key share $z_i = \sum_{j=1}^{n} s_{j,i}$.

Pedersen proposed this protocol for an honest majority, that is, $n \geq 2k-1$, and this is the model adopted by Verificatum [24]. However, Cortier et al. offer a proof that, when the protocol is used in combination with ElGamal encryption, it offers IND-CPA security under the DDH assumption for arbitrary threshold

choices, that is, for any $k \leq n$ [9]. (The other reference [6] mentioned in the Belenios specification [13] focuses on the one-round non-threshold protocol.)

Pedersen's DKG is insecure in the case of a dishonest majority. As discussed above, a central aspect of the security of a DKG is to make sure that no subset of less than k trustees can "force" the public key to take a value of their choice, of which they would know the corresponding discrete logarithm. For the single-round protocol (and of the ElectionGuard protocol), this is achieved by forcing every trustee to offer a Schnorr proof that it knows the s_i value corresponding to its public key K_i. For the Pedersen protocol, this is achieved by having at least k trustees verify the correctness of the shares they received at Step 3 of the protocol: if k trustees hold correct shares of $a_{i,0}$, then it must be the case that T_i knows $a_{i,0}$. The assumption of an honest majority guarantees that at least k trustees will perform the expected verification steps. However, when there is a dishonest majority, this is not the case anymore (this is overlooked in [9]), and a dishonest trustee might be able to simulate the VSS steps for the honest trustees, while ignoring its secret key share s_i, opening the way for arbitrarily choosing the final public key K.

Such attacks have been known for a long time (see Langford [16] for instance) and, to make it concrete, we illustrate the process in the simple case where $n = k = 3$. Let us assume that T_1 is honest and that T_2 and T_3 would like to be in full control of the election public key K. T_1 selects $P_1(x)$, publishes $(K_{1,0}, K_{1,1}, K_{1,2})$, then sends $P_1(2)$ and $P_1(3)$ to T_2 and T_3 as per the protocol definition. We let T_2 follow the protocol as well. Then T_3 selects $s \leftarrow \mathbb{Z}_q$, sets $K = g^s$ and $K_{3,0} = K/(K_{1,0}K_{2,0})$ so that $K = \prod_{i=1}^{3} K_{i,0}$. T_3 then selects a random $s_{3,1} \leftarrow \mathbb{Z}_q$ to be sent to T_1, a random $K_{3,1} \leftarrow \mathbb{G}$ and sets $K_{3,2} = g^{s_{3,1}}/(K_{3,0}K_{3,1})$ so that the share verification equation $g^{s_{3,1}} = \prod_{\ell=0}^{2}(K_{3,\ell})^{(1^\ell)}$ is satisfied. At this stage, the honest trustee T_1 is satisfied and accepts K as the election public key, despite the fact that T_3 knows the corresponding secret key. The full protocol transcript is distributed just as a normal protocol execution. This attack can be generalized to any situation where the honest trustees do not receive enough shares to reconstruct the secrets of the corrupted trustees. The crucial step is the generation of the proper $K_{i,j}$ values that are consistent with the shares sent to the honest trustees, but guaranteeing that the verification equations are satisfied can always be achieved by Lagrange interpolation "in the exponent".

Common mitigations include an initial round during which every trustee commits to its $K_{i,j}$ values before opening them and resuming the protocol, or requiring every trustee to provide a Schnorr proof that it knows the discrete logarithms of its $K_{i,j}$ values w.r.t. g. The second option is the one adopted in ElectionGuard [4].

The DKG in Belenios. Fortunately, Belenios does *not* exactly follow [9] but does something slightly different: it requires each trustee T_j to offer a proof of knowledge of z_j defined as the logarithm in base g of $g^{z_j} = \prod_{i=1}^{n} g^{s_{i,j}} =$

$\prod_{i=1}^{n}\prod_{\ell=0}^{k-1}(K_{i,\ell})^{j^\ell}$. To the best of our knowledge, this option is original, and we argue here that it offers security as well.

We use the name Belenios ElGamal for the encryption scheme that uses the Pedersen DKG augmented with a Schnorr proof of knowledge of z_j provided by each trustee as its key generation protocol and encrypts with ElGamal encryption. For simplicity, we assume here that there are secret and authentic communication channels available between all pairs of trustees for communicating the shares and that trustees verify that they have a common view of the protocol public elements.

Theorem 1. *Belenios ElGamal encryption is IND-CPA secure under static corruption of up to $k-1$ trustees ($1 \leq k \leq n$) if standard ElGamal encryption (with standard single-party key generation) is IND-CPA secure with the same public parameters.*

Proof. Let k and n be fixed and define $C = \{1, \ldots, k-1\}$ and $H = \{k, \ldots, n\}$. Assume, w.l.o.g., that the trustees in the set $\{T_i\}_{i\in C}$ are corrupted, while the others are honest. We design an adversary B against standard ElGamal encryption that wins the IND-CPA game with a probability equal, up to a negligible difference, to the probability that an adversary A controlling the corrupted trustees breaks the IND-CPA security of Belenios ElGamal encryption.

Given A, we design B as follows: when B receives the ElGamal public parameters $(\mathbb{G}, q, g)$ and the public key K^* from the standard ElGamal challenger, it forwards the public parameters to A. B honestly plays the role of the honest trustees, except for T_n. For emulating T_n, it simulates the sharing of the discrete logarithm of $K_{n,0} = K^*$ by picking $s_{n,i}$ as a random element of $\mathbb{Z}_q$ for every $i \in C$. It then derives $\{K_{n,i}\}_{i\in C}$ so that $g^{s_{n,i}} = \prod_{j=0}^{k-1}(K_{n,j})^{i^j}$ for every $i \in C$, by Lagrange interpolation "in the exponent" – this works because $|C| < k$. The view of A is distributed in a way that is identical to what it would be in a normal execution of the Belenios ElGamal key generation, and A completes the protocol on behalf of the corrupted trustees. When B received all its shares from the corrupted trustees, it can program the random oracle in order to produce simulated Schnorr proofs of knowledge of z_i as the logarithm of $\prod_{j=1}^{n} g^{s_{j,i}} = \prod_{j=1}^{n}\prod_{\ell=0}^{k-1}(K_{j,\ell})^{i^\ell}$ in base g for every $i \in H$. B checks the resulting transcripts and fails if anything is wrong.

If all verifications succeed, B can now extract the $z_i = \sum_{j=1}^{n} s_{j,i}$ values for $i \in C$ from the Schnorr proofs provided by A. Substracting the shares that it sent on behalf of the honest trustees, A can also compute $z_{C,i} = \sum_{j\in C} s_{j,i}$ for $i \in C$. When $i \in H$, the values $z_{C,i} = \sum_{j\in C} s_{j,i}$ can also be computed, since the $s_{j,i}$ values are shares that have been sent by A. As a result, A has all n shares of $s_C = \sum_{j\in C} s_{j,0}$, and can reconstruct that value.

Eventually, when A asks for the encryption of a pair of messages (m_0, m_1), B forwards it to the ElGamal challenger, who returns a ciphertext $(c_0, c_1) = (g^r, m_b(K^*)^r)$. B then submits to A the ciphertext $(c_0, c_1(c_0)^{s_C+\sum_{i=k}^{n-1} s_i}) = (g^r, m_b K^r)$ When A outputs a guess b' on b, B forwards that guess to the ElGamal challenger. The probability that $b = b'$ is exactly the one that A makes a

correct guess in the Belenios ElGamal IND-CPA security game. The only possible discrepancy comes from the potential failures to simulate or extract a Schnorr proof, which can be made negligible.

3 Protocol setups

The above protocols may be cryptographically correct, but could be completely useless if their setup assumptions are not satisfied, or if it is possible to completely circumvent them. For instance, trustees need a mechanism to verify that the key used in the election is really the one they generated in the DKG protocol and has not been replaced with a key that is fully controlled by another entity.

Elections, whether they are electronic or not, cannot exist in isolation. They require a service that provides a public bulletin board of some form. Fundamentally, voters need access to an authentic source of blank ballots, they need to know where and when to cast their votes, and where to read the election results. We simply assume that a public bulletin board is available—how to build bulletin boards has been largely discussed in other places, see for example [11,15].

The single-key and one-round DKG protocols can directly be implemented when a bulletin board is available: trustees must verify that the public key they produced has been posted on the bulletin board, and that it has not been replaced by the bulletin board manager for instance. Election verifiers can check that all ballots are encrypted and then tallied w.r.t. the correct public keys.

The threshold DKG protocol, however, additionally requires secret and authentic communication channels between trustees to exchange key shares. The assumption that such channels exist is standard and appears virtually everywhere in the DKG literature (see, e.g., [18,12]). Such channels are easy to implement using encryption and signature mechanisms (e.g., via TLS) assuming a trusted public-key infrastructure (PKI).

However, in the election context, to make the assumption that a trusted PKI exists and that trustees possess certified keys, is challenging. Trustees will rarely have a certified signature key and, depending on the context, the distributed trust provided by a threshold DKG might be undermined by relying on certificates signed by a single centralized authority, which in turn may be external to the election process and be driven by different incentives.

In practice, this is addressed in various ways. We describe a few examples from systems that use a threshold DKG.

1. In Verificatum, each trustee generates signing keys and gives the corresponding verification keys to the other trustees. It is suggested that, "in practice, the operators could organize a physical meeting to which they bring their laptops and execute the above steps" and "for convenience, hexadecimal encoded hash digests of files can be computed using vmni to allow all parties to check that they hold identical protocol info files at the end" [22, p. 2].
2. In Belenios, each trustee generates signing and encryption keys, which are shared with the other trustees via the voting server. At the opening of the

election, each trustee "checks that [its own certificate] appears in the set of verification keys PK of the election" [13, p. 5].

3. In ElectionGuard, trustees do not generate signing keys. Instead, at the conclusion of the DKG, a preliminary record that contains all public information from the DKG execution is published on the bulletin board. It includes all encryption keys used to exchange shares, the $K_{i,j}$ values, and matching Schnorr proofs. Trustees must hash this information and compare it to a hash computed from their own view of the DKG execution [4, p. 27].

Verificatum and Belenios create authentic channels using signing keys. Verificatum suggests direct contact between the trustees, allowing direct comparison and confirmation of all the keys that they are going to use. Of course, it remains important for the trustees to verify that the correct key material is also published as part of the official election record on the bulletin board. Belenios focuses on direct verification of the election record and that trustees confirm the presence of their own signing (and encryption) keys. We pointed out that trustees should agree on all the keys that have been used in order to ensure that key shares are not compromised. ElectionGuard authenticate the view of the protocol execution rather than signing keys used to sign that view. This has essentially the same effect when verification is successful. Of course, signing keys may additionally provide a non-repudiation property, which could be used for accountability should problems occur. However, since signing key generation is part of the protocol, a malicious trustee might as well complain that the signature verification key published on its behalf is incorrect. Here, the in-person protocol suggested in Verificatum may be advantagous when trustees are required to agree in-person on their keys, that is, in a setting where authenticity cannot be questioned. It is also the most demanding option for human trustees.

4 Trustees, their hardware, and their software

The protocols in Section 2 guarantee their expected cryptographic security notions. However, as we have seen in the previous section, their security cannot be guaranteed by machines *only*: the machines running the protocols need to be bound to a specific context and, if one relies on a designated set of trustees, these trustees must confirm authenticity of the key material used in the election.

These challenges become considerably more pronounced when organizing an election with trustees in practice. Trustees need to run software on actual hardware. We elaborate on these issues in this section.

4.1 Key generation ceremonies in practice

We start our discussion by reviewing some of the reported approaches in which the above key generation protocols have been deployed, focusing on the choice of trustees, software, and hardware that were made, even if the information is often scarce.

Estonia. At least in the early, pre-IVXV version of their Internet voting system, Estonia used an HSM to generate a single key and decrypt all ballots after a threshold of election officials inserted their "cryptosticks" [8].

We do not know what is offered for verification here, but we assume that the HSM can produce an attestation that it generated the election key and that the key was never released, as well as a list of all the ciphertexts that were decrypted using that key over its life time.

The task of trustees is highly limited here: vote secrecy essentially depends on proper anonymization of the encrypted votes before they are decrypted (which may not be an obvious operation in the absence of the verifiable mixnet that was introduced in a later version of the system), and on verifying that the HSM has not been abused to perform unauthorized decryption or key export operations.

UC Louvain. Since version 2.0, Helios uses the one-round distributed key generation described above. To the best of our knowledge, it has not been deployed in government elections, but we have a description of at least one deployment for a university election at UCLouvain [1].

In that election, trustees were selected from various voter groups and worked with the help of external experts. An in-person meeting was organized for the key generation. Trustees were provided with laptops from which hard-disk drives and network interfaces had been removed. The laptops were booted on Linux using live-CDs, and minimal Python code was provided to generate the election keys, under control of the external experts. Public and secret keys were generated and saved on multiple USB sticks (including copies of the secret keys as backup). The laptops were then turned off and the CDs destroyed. Copies of the public keys were uploaded in the system, and trustees were asked to compare the public keys published on their behalf with their own copies.

So, efforts were made to restrict the ways in which secret keys could be exfiltrated from the hardware, but such measures obviously require expert control.

The software used by the trustees apparently was provided by the same source, but it seems that this code was simple enough to be reviewed during the key generation process—again, this requires expert control. Throughout, trustees remained in control of their secret keys. The potential loss of any one of these keys could have prevented the election organizers from tallying the election—hence the existence of backup copies.

We can see that there are tensions between different incentives here. On the one hand, the cryptographic protocol is designed to put the trustees in control of their keys. On the other hand, should a trustee be missing or lose a key, the tallying process might fail. In such circumstances, the blame is likely to be put more strongly on the election organizer and technology supplier than on the citizen who volunteered to help as a trustee. This actually places a strong incentive on the organizer and technology provider to "tweak" key generation in a way that allows them to obtain copies of all the keys, not to break the privacy of the votes, but to recover from a human failure by a trustee that would badly reflect on them.

Switzerland. The hardware and roles for key management in the Swiss Post voting system used for Swiss government elections is publicly available [20].

Its DKG is a variation of the one-pass protocol above, with some keys generated by Swiss Post and other keys generated by the cantons organizing elections.

Swiss Post holds four keys, generated on four control components, running four different hardened operating systems in order to mitigate against OS-level exploits (Debian, RedHat, Ubuntu, and Windows are listed). The cantons use multiple laptops deployed in a secure offline environment: they receive the election data through USB sticks.

At the Swiss Post level, the operations are under control of multiple expert teams, even though all of them seem to work under the authority of Swiss Post. The security operations and human resources at the canton level are less documented, and we can guess that they vary among cantons.

At least at Swiss Post, the trustee tasks are performed by experts, but with limited independence. The generation of other keys at the Canton level may offer an important level of human independence though. The level of independence of software and hardware is not clear: if the voting software, OS, and hardware are provided by Swiss Post, then the effective independence may be reduced. Again, we do not know how this is handled at the canton level.

Franklin County, Idaho. Hardware and key management in a deployment of the ElectionGuard SDK during the 2022 General Election in Franklin County, Idaho is documented in [17]. ElectionGuard uses a threshold DGK protocol, i.e., trustee devices had to communicate with each other during key generation.

Trustee devices were connected to an administrator device on a local network. Trustees were ordinary citizens who operated devices that they were given, running pre-installed software. At the end of key generation, the trustee devices storing their private keys were kept in safes controlled by the election administrator. The devices themselves required the trustee fingerprint to be activated. However, in order to recover from device failures (and despite the use of a threshold DKG), keys were also exported on thumb drives and stored in the same safe.

The real impact the trustees had here seems minimal. Trustees had no control of the software and hardware, and even the keys were kept in the custody of the election administrator. However, one should not expect that the expertise that can be deployed to run a national voting system in Switzerland can also be deployed in much smaller elections: the election records show that 113 ballots were cast and 2 were spoiled in that election. In that specific deployment, no linkage between the individual paper ballots and the voters who cast them was maintained; so encrypted ballots have a much weaker link to the voter identity than in the Internet voting systems described above, in which a voting server controls the identity of the voter and receives its encrypted ballot.

Summing up. It is clear that, in all these cases, keys that safeguard vote privacy hardly are in the sole control of trustees that are representatives of the general population concerned by the election.

In all cases, the systems rely on authorities to provide the trustees with experts and trustworthy hardware and software. Even though the entire purpose

of splitting keys amongst independent entities is to prevent a curious central authority from simply reading votes, in practice, there is little to stop a curious authority providing software that reveals keys or performs additional decryptions. Even if the software is correct, it is unlikely that anyone will notice if a curious authority simply asks for additional decryptions.

Privacy of votes is ensured if the cryptographic assumptions underlying the encryption system hold *and* if a sufficiently large set of trustees perform their assigned tasks properly. But the trustees themselves rarely know whether or not they have performed their tasks properly, and have few practical ways of making sure that they did. So how are observers expected to develop confidence?

Trustees charged with generating and managing keys should be well-trained and thoroughly understand their role. They should bring their own software and devices—obtained from sources they choose to trust. They should inspect the data they are asked to decrypt and ensure that it contains only the values necessary and appropriate to complete the election process. In theory, this is great; but from what we see in practice, we are still far from there.

5 The Hardware Alternative

While audit data can be published in order to guarantee the integrity of election results independently of any requirement to trust designated parties (trustees, election administration, etc.), the solutions we described above, even if they were deployed perfectly, require trusting that a subset of a fixed group of trustees are behaving correctly, and this behavior includes correct human behavior (humans do not distribute copies or misuse their keys) but also correct software and hardware (the software and the hardware do not leak or misuse secret key material).

Hardware-based security technologies, while existing for a long time, have made tremendous advances during the last few years. The capabilities of hardware security modules (HSMs) dramatically expanded, and trusted execution environments (TEEs, including Intel SGX and AMD SEV-SNP) are now readily available. These technologies open the opportunity to replace vague and difficult to verify assumptions on human processes with clear, specific, and independently-verifiable assumptions on hardware.

HSMs and TEEs can generate keys, log their use, delete the keys after their use, and publish signed attestations of all of these steps. Independent observers can verify the attestations made by these devices and the certificate chain to the device manufacturers. Of course, it is possible that a device does not perform as advertised [3,21] or that the certificate chain to a manufacturer is compromised. But the assumptions required here need to be contrasted with the assumptions traditionally made of humans who may have little understanding of the processes. And, again, the risks can be mitigated thanks to the threshold key management techniques discussed above.

5.1 Secure hardware technologies

Hardware Security Modules. Hardware security modules (HSMs) have been used for decades to provide physical protection for high-value keys that do not need to be used often. They offer tamper-resistance and a variety of means for an authorized set of users to request actions be taken such as generating a key pair, exporting a public key, and using a protected private key to sign or decrypt given data. While these actions are often limited by default to a specific set of cryptographic algorithms that may not be compatible with the type of DKG protocols described above, custom firmware can often be loaded, and high-end HSMs even offer enclaves that support arbitrary code execution.

These features nicely match the requirements of an election. A key may be generated a month or more before any decryptions are required, and decryptions can be done together in batch in a single session. The principal drawback of HSMs is their operational complexity and high cost, especially if they need to offer the flexibility and performance needed to execute DKG protocols.

Secure Enclaves. Secure enclaves are a newer technology offered, for example, by Intel's SGX and AMD's SEV-SNP technologies. They are capable of executing arbitrary code and providing attestations about the code they ran and the results produced, while offering strong isolation and encrypting all their communication with memory external to the CPU.

Although they are more flexible and far less expensive than HSMs, secure enclaves offer less robust physical protections, as evidenced by an already long history of side-channel vulnerabilities [21]. Another disadvantage of secure enclaves is that they only live as long as their host devices are powered and running. There are some means for preserving and reconstituting state, but doing so enables rollback attacks which can allow keys to be used without their use being logged. Therefore, to be useful for elections, a single secure enclave would need to be sustained for the entire period from key generation through results decryption. Generic solutions aiming at detecting rollbacks are being explored [2], but we will explore more direct options below.

Trusted Platform Modules. Trusted platform modules (TPMs) are security components available in most commercial PCs. Their principal utility is to provide externally-verifiable attestation of the software running on a PC, and they have been successfully deployed to secure open source voting machines [23]. The principal difference between the secure enclave approach and TPMs is that TPMs offer no protection of the data being computed on while secure enclaves offer protections to make it more difficult to access data externally. This makes TPM technology less attractive for handling the role of a trustee.

5.2 Resilient Usage of Secure Hardware

We now outline how secure enclaves, either permanent on an HSM, or volatile in a TEE, can be leveraged to offer an alternative to human trustees.

The starting point is to deploy threshold encryption and decryption protocols on a set of secure enclaves running on multiple devices. Each enclave runs open-source trustee code, and offers an attestation that the expected code is running. The attestation can be tied from the specific hardware component to its manufacturer.

The enclaves are coordinated by an election administrator. The administrator submits inputs to the enclaves and collects their outputs. Crucially, the administrator is required to publish attestations of all the operations performed by the enclaves. A trustee protocol will be defined as secure if the publication of valid attestations produced by the enclaves guarantees the expected security properties, that is, the only decryption operations that will ever be performed w.r.t. the election public key are those attested to by the enclaves.

Intuitively, if the hardware is trusted, a single enclave would be enough. However, we are concerned that the single enclave might fail, and the primary reason for using multiple enclaves is now to offer resilience to hardware failures, which is very different from the primary goal in the context of human trustees, which was to guarantee that a single trustee cannot silently decrypt ballots. Of course, running enclaves on a heterogeneous set of devices from a variety of vendors also offers additional privacy assurances, should a specific device fail to guarantee the isolation and attestation properties expected.

The code starts with a round of setup: each enclave generates a signature key pair and submits the signature verification key to the administrator, who returns an election identifier `eid` and the signature verification keys provided by the other enclaves. Each enclave attests to the list of signature verification keys that it is going to be using in the context of election `eid`. The signature keys are used to establish secure communication channels between the enclaves.

From this moment we can rely on a set of protocols, which all exist and are described by Chen and Lindell for instance, who also prove their security [7].

1. *Distributed key generation.* This protocol runs a threshold DKG with a quorum of k enclaves within a set of n, based on n parallel executions of Feldman's VSS protocol.
2. *Refresh.* This protocol refreshes existing shares of a given secret, essentially by adding to the existing shares a set of freshly generated shares of 0.
3. *Removing a device.* This protocol makes it possible to invalidate the share held by a specific enclave, essentially by running a refresh of the shares while excluding that enclave from the refresh. At the end of this protocol, the quorum of k enclaves is maintained, but only $n-1$ enclaves hold shares.
4. *Adding a device.* This protocol makes it possible to add a new enclave, resuming the level of robustness from a set of $n-1$ enclaves to n enclaves (without any change in the quorum k).

The election administrator starts by triggering the execution of the DKG protocol. At the end of this protocol, each enclave attests to its public view of the execution and to the resulting public key in particular. Furthermore, each enclave also attests to the secure deletion of all the key shares it saw, except for the single one that needs to be kept for future decryption operations.

On a regular basis, the election administrator starts a refresh protocol. This provides proactive security [14] and protects against an adversary who would manage to extract key shares from an increasing number of enclaves (e.g., by successfully running side-channel attacks). Each refresh resets all the shares and renders the extracted shares useless. This does not need to reflect on decryption proofs, which can be made share independent [4].

At any point in time, enclaves might be identified by the administrator as failing – be it rightly so, as a result of a power failure for instance, or in order to maliciously exclude the enclave. The purpose of such an exclusion might be to elude the requirement for the enclave to attest the destruction of its stored key shares, hence weakening the privacy of the votes. However, any such failure statement is required to be followed by an execution of the enclave removal protocol by the other enclaves, which will effectively render useless the share of the excluded enclave.

Of course, such an exclusion reduces the resilience of the system to the failure of other enclaves: one may progressively reach a state in which more than $n - k$ enclaves have been removed, preventing any further operation. In order to avoid this situation, a new secure enclave must be started, its signature verification key distributed, and the enclave addition protocol must be executed.

Whenever tallying operations start, the election administrator submits the required decryption requests to the enclaves, who simply decrypt whatever they are asked to decrypt without any specific verification – which they could not perform without any visibility of the ballots submitted in the election. However, each decryption operation is securely logged by each device.

When the election is complete, the election administrator requires each device to erase its secret key share. Each device eventually provides an attestation of all the decryption operations that have been performed until its final key shares have been erased, and the final erasure is confirmed.

All the attestations produced by the devices are now published for auditing. The core benefit of this approach is that, if the audit succeeds, and under the assumption that a quorum of the devices on which the enclaves are running offer the advertised security guarantees, then we obtain guarantees on the secrecy of the votes. In effect, we introduce privacy verifiability in elections.

References

1. Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In *2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '09*. USENIX Association, 2009.
2. Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath T. V. Setty, and Sudheesh Singanamalla. Nimble: Rollback protection for confidential cloud services. In *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023*, pages 193–208. USENIX, 2023.
3. Jean-Baptiste Bedrune and Gabriel Campana. Everybody be cool, this is a robbery! In *IACR Real World Crypto*, 2020.

4. Josh Benaloh, Michael Naehrig, and Olivier Pereira. ElectionGuard design specification version 2.1.0. https://www.electionguard.vote/spec/, May 2024.
5. Josh Benaloh, Michael Naehrig, Olivier Pereira, and Dan S. Wallach. Electionguard : a cryptographic toolkit to enable verifiable elections. In *USENIX Security'24*, 2024.
6. David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 626–643. Springer, 2012.
7. Yi-Hsiu Chen and Yehuda Lindell. Feldman's verifiable secret sharing for a dishonest majority. Cryptology ePrint Archive, Paper 2024/031, 2024. https://eprint.iacr.org/2024/031.
8. Dylan Clarke and Tarvi Martens. *Real-world Electronic Voting: Design, Analysis and Deployment*, chapter E-Voting in Estonia, pages 129–141. CRC Press, 2017.
9. Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Distributed ElGamal à la Pedersen: Application to Helios. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013*, pages 131–142. ACM, 2013.
10. Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. Belenios: A simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows*, volume 11565 of *LNCS*, pages 214–238. Springer, 2019.
11. Chris Culnane and Steve A. Schneider. A peered bulletin board for robust use in verifiable voting systems. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014*, pages 169–183. IEEE Computer Society, 2014.
12. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol.*, 20(1):51–83, 2007.
13. Stéphane Glondu. Belenios specification. https://www.belenios.org/specification.pdf. Version 2.5.
14. Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology - CRYPTO '95*, volume 963 of *LNCS*, pages 339–352. Springer, 1995.
15. Lucca Hirschi, Lara Schmid, and David A. Basin. Fixing the achilles heel of e-voting: The bulletin board. In *34th IEEE Computer Security Foundations Symposium, CSF 2021*, pages 1–17. IEEE, 2021.
16. Susan K. Langford. Weakness in some threshold cryptosystems. In *Advances in Cryptology - CRYPTO '96*, volume 1109 of *LNCS*, pages 74–82. Springer, 1996.
17. Microsoft. End-to-end verifiability in real-world elections. https://www.electionguard.vote/images/EAC%20Report%20Final.pdf, January 2023.
18. Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *Advances in Cryptology - EUROCRYPT 1991*, volume 547 of *LNCS*, pages 522–526. Springer, 1991.
19. Swiss Post. Cryptographic primitives of the swiss post voting system. https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives, February 2024.
20. Swiss Post. E-voting architecturedocument. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/raw/master/System/SwissPost_Voting_System_architecture_document.pdf, February 2024.

21. Stephan van Schaik, Alex Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Christina Garman, Daniel Genkin, Andrew Miller, Eyal Ronen, and Yuval Yarom. SoK: SGX.Fail: How stuff get eXposed. In *IEEE S&P Symposium*, 2024.
22. Verificatum. User manual for the verificatum mix-net. https://www.verificatum.org/files/vmnum-3.1.0.pdf, September 2022.
23. VotingWorks. Install.md. https://github.com/votingworks/vxsuite-complete-system/blob/main/INSTALL.md, November 2022.
24. D. Wikström. Verificatum. https://www.verificatum.org/, May 2022.