Secure Voter Identification With Short Keys

No Author Given

No Institute Given

Abstract. For submitting electronic votes over the Internet, entering sufficiently long cryptographic keys on a keyboard is not a very practicable means of identification. In places without an established public-key infrastructure, current solutions are usually making compromises on either the security, the verifiability, or the usability of the system. In this paper, we present a new pairing-based identification protocol, which offers the desired cryptographic security, ~~but with secret keys as short as the security parameter~~. Keys of that size are much more likely to be accepted by voters, if entering them is only required occasionally, for example once a year on the election day or less. Our proposed scheme allows the keys to be generated in a distributed manner, for example in a setting where voters obtain their credentials from the authorities without registration. In the aftermath of an election, our scheme enables third parties to verify that only votes from eligible voters have been counted.

The proof presented in this paper draft is incomplete, so this statement was not fully proven.

Indeed, the statement is false, as there is a counter-example to the missing part of the proof.

While the identification scheme as a whole seems to be secure, its secret key length is comparable to that of other similar, less complex schemes.

Keywords: Voter identification · eligibility verifiability · identification protocols · Fiat-Shamir signature schemes · pairing-based cryptography.

1 Introduction

The ability to identify eligible voters reliably is fundamental for an e-voting system to avoid ballot stuffing and other forms of impersonation attacks. Along with end-to-end verifiability, systems today are also expected to offer *eligibility verifiability*, meaning that any third party can check that only votes from eligible voters have been counted. This is something that frequently used identification protocols based on user names and passwords cannot offer. Besides, simple password-based protocols can be broken too easily by an attacker intercepting the password transmission or brute-forcing a stolen password database. High-risk applications like e-voting certainly require much stronger methods of user identification.

The most obvious secure way of ensuring eligibility verifiability would be to establish a public-key infrastructure (PKI) that covers the entire electorate. Voters could then publish their digitally signed ballots on the public bulletin board, and everyone could verify that only votes from eligible voters have been counted by checking the attached signatures. While participation privacy is difficult to achieve in this approach, vote privacy can be established with mix-nets, homomorphic tallying, or other techniques. One problem with a PKI-based approach is the management of the secret keys, which are typically at least twice as long as the security parameter (for example 256-bit keys for 128 bits security).

Since memorizing and entering such keys cannot be expected from average users, they are usually stored on smart cards or similar hardware tokens. In many places of the world, unfortunately, such infrastructures are not yet available.

1.1 The Swiss E-Voting Approaches

The approach presented in this paper is motivated by current system proposals in Switzerland. For political elections, there are two competing approaches. Each of them must deal with the problem that currently no PKI exists for identifying the voters, for example in form of a nationwide eID system.Therefore, personal voting credentials are sent in printed form to the voters over postal mail in the run up to every election, together with other printed materials such as the official election brochure or the verification code sheet. Postal mail is considered a trustworthy channel.

In the Swiss Post voting system [2], the printed credentials consist of two secrets, a 20-character alphanumeric password (called *start voting key*) and a 9-digit numeric password (called *ballot casting key*). Technically speaking, the former is a PBKDF2 password for deriving the *verification card secret key* (an element of $\mathbb{Z}_q$ for $||q|| \in \{2047, 3071\}$), while the latter is a secret parameter of a distributed function for deriving the *short vote cast return code* (a 8-digit number). Both parts of the credential are generated and printed by a single fully trusted party called *print office*. In total, voters must enter approximately 130 bits of entropy to cast a vote, independently of the chosen security level of either 112 bits (default security) or 128 bits (extended security). Given that the credentials are not used for signing the submitted ballots, the Swiss Post voting system does not offer eligibility verifiability to third parties according to common definitions from the literature [7, 11].

In CHVote [5], the printed credentials also consist of two secrets. They are both secret keys in the setting of a Schnorr identification protocol, i.e., for achieving σ bits of security, they are elements of $\mathbb{Z}_q$ for $||q|| = 2\sigma$. The keys are generated in a distributed manner by a group of s authorities. A fully trusted *printing authority* assembles the key shares and prints them on paper. In total, voters must enter $2||q|| = 4\sigma$ bits of entropy to cast a vote, for example 448 bits (112 bits security) or 512 bits (128 bits security). By instantiating the Schnorr identification protocol in form of an non-interactive signature scheme, CHVote offers unrestricted eligibility verifiability based on the information published during a protocol run.

In comparison with the Swiss Post voting system, CHVote is based on weaker trust assumptions, offers a stronger form of authentication, and provides better verifiability. However, as voters are required to enter considerably longer keys, it strongly restricts the usability of the vote casting process. Offering the same security and verifiability under comparable trust assumptions, but with much shorter keys, is the main motivation for this paper.

1.2 Usability Considerations

If achieving better usability with shorter keys is the main goal of this paper, we may ask the question of the maximal key length that voters may accept when entering the key over the keyboard.[1] Clearly, careful usability studies would be necessary to answer this question conclusively, but conducting such studies or summarizing the results of existing studies is outside our area of expertise and beyond the scope of this paper. Nevertheless, in order to provide some background information on this important topic, we shortly discuss the problem of encoding cryptographic keys in form of corresponding strings of characters. Clearly, the length of these strings strongly depends on the size of the chosen alphabet, from which the characters are taken.

Table 1 gives an overview of alphabets commonly used for such encodings. While large alphabets lead to shorter encodings, they also reduce the user's typing speed. Furthermore, since entering randomly looking characters by users is an error-prone process, it is desirable that the chance of misspellings is as small as possible. Case-insensitive alphabets and *fail-safe* alphabets not containing homoglyphs such as `'0'` and `'O'` are therefore preferred. The presence or absence of these properties in the considered alphabets is indicated in Table 1. The table also shows the entropy of a single character in each alphabet (measured in bits per character).[2]

Assuming that $\sigma \in \{112, 128\}$ bits of security are currently the most common choices, we have computed the length of corresponding encodings in different alphabets. Since fail-safe alphabets are preferable in practice, the results shown in Table 2 are restricted to those. To detect errors made during typing with sufficiently high probability, we also consider encodings of keys that contain 10 percent of redundant checksum bits (except for A_{7776}, which already contains a sufficient amount of redundancy). The upper part of Table 2 contains entries for keys with 2σ bits of entropy, and the lower part the entries for keys with σ bits of entropy. The shown values indicate that keys of length σ are much more acceptable for users than keys of size 2σ. In the rest of this paper, we will call them *short keys*.

1.3 Contributions and Overview

We propose in this paper a new identification protocol ~~with short secret keys that are just as long as the number of desired security bits σ~~. Compared to existing

[1] As an alternative to entering keys over the keyboard, one could think of encoding them into a QR-code. While scanning QR-codes gets more and more popular in many application areas, it imposes additional requirements regarding the equipment available on the voter's side. In this paper, we assume that a keyboard is the only input device available to everyone.

[2] A special case in Table 1 is the entry for the *Diceware Wordlist*, which consists of $6^5 = 7776$ English words. Since human users are well-trained in entering words in a natural language, entering a sequence of Diceware words is clearly less error-prone than entering codes consisting of random characters. Note that similar wordlists are available in many different languages.

Name	Alphabet	CI	FS	Bits/Char
Decimal	$A_{10} = \{\texttt{0}, \ldots, \texttt{9}\}$	●	●	3.32
Hexadecimal	$A_{16} = \{\texttt{0}, \ldots, \texttt{9}, \texttt{A}, \ldots, \texttt{F}\}$	●	●	4
Latin	$A_{26} = \{\texttt{A}, \ldots, \texttt{Z}\}$		●	4.70
Alphanumeric	$A_{32} = \{\texttt{0}, \ldots, \texttt{9}, \texttt{A}, \ldots, \texttt{Z}\} \setminus \{\texttt{0}, \texttt{1}, \texttt{I}, \texttt{O}\}$	●	●	5
	$A_{36} = \{\texttt{0}, \ldots, \texttt{9}, \texttt{A}, \ldots, \texttt{Z}\}$	●		5.17
	$A_{57} = \{\texttt{0}, \ldots, \texttt{9}, \texttt{A}, \ldots, \texttt{Z}, \texttt{a}, \ldots, \texttt{z}\} \setminus \{\texttt{0}, \texttt{1}, \texttt{I}, \texttt{O}, \texttt{l}\}$		●	5.83
	$A_{62} = \{\texttt{0}, \ldots, \texttt{9}, \texttt{A}, \ldots, \texttt{Z}, \texttt{a}, \ldots, \texttt{z}\}$			5.95
Base64	$A_{64} = \{\texttt{A}, \ldots, \texttt{Z}, \texttt{a}, \ldots, \texttt{z}, \texttt{0}, \ldots, \texttt{9}, \texttt{=}, \texttt{/}\}$			6
Diceware[3]	$A_{7776} = \{\texttt{"abacus"}, \ldots, \texttt{"zoom"}\}$	●	●	12.92

Table 1: Common alphabets with different sizes and characteristics. Case-insensitivity (CI) and fail-safety (FS) are desirable properties for reducing the likelihood of incorrect user entries.

~~approaches such as Schnorr's identification protocol [10], the secret keys in our protocol are at least 50% shorter for achieving the same level of security. Relative to Okamoto's witness-hiding identification protocol [9], our secret keys are even 75% shorter.~~

We prove that our protocol is secure against passive attacks, which implies that it can be transformed into a secure signature scheme using the classical Fiat-Shamir construction [1]. Furthermore, just like in the protocols of Schnorr and Okamoto, keys can be aggregated by simply applying the corresponding group operation. Key pairs can therefore be generated in a distributed matter by a group of parties of which only one needs to be trusted. Our protocol is based on bilinear pairings, which makes the verification algorithm less efficient in comparison with existing protocols. As mentioned earlier, the primary use case of our new protocol are the current Swiss e-voting protocols, but we believe that our approach is cryptographically interesting in its own right.

In Section 2, we introduce the notion of an identification protocol and summarize the most common security definitions. We also discuss the Fiat-Shamir construction of a secure signature scheme from an identification protocol and its application to e-voting. In Section 3, we present our new pairing-based identification protocol and compare it to existing approaches. Section 4 provides a formal security proof for our protocol, and Section 5 concludes the paper.

2 Identification Protocols

This section gives a general introduction to identification protocols and their security properties. We adopt most definitions from Boneh and Shoup's lecture notes [1], which give an excellent survey of the topic. We also discuss the relation

Key Length	Entropy	Checksum	A_{10}	A_{16}	A_{26}	A_{32}	A_{57}	A_{7776}
282	256	26	85	71	60	57	49	–
256	256	–	78	64	55	52	44	20
247	224	23	75	62	53	50	43	–
224	224	–	68	56	48	45	39	18
141	128	13	43	36	30	29	25	–
128	128	–	39	32	28	26	22	10
124	112	12	38	31	27	25	22	–
112	112	–	34	28	24	23	20	9

Table 2: Lengths of key encodings for different key lengths, optional checksum bits, and commonly used alphabets.

between identification protocols and signature schemes, which is important for applying the protocol presented in this paper to the e-voting use case.

2.1 Basic Definitions

An *identification protocol* is an interactive two-party protocol to be carried out between a *prover* and a *verifier*. Their computations are modeled in terms of interactive protocol algorithms P and V, respectively. We write $P(sk)$ for the initialization of the prover's state_P with a *secret key* sk and $V(vk)$ for the initialization of the verifier's state_V with a *verification key* vk. These keys are obtained from executing a probabilistic *key generation algorithm* G, which takes no input and outputs a pair (sk, vk). By executing the protocol, P tries to convince V of its identity by proving knowledge of sk, i.e., at the end of the interaction between P and V, V outputs either accept or reject, depending on whether P's identity claim has been confirmed. If $V(vk)$ outputs accept with probability 1 after interacting with $P(sk)$, for all possible pairs $(sk, vk) \leftarrow G()$, then the triple $\mathcal{I} = (G, P, V)$ is called an identification protocol.

Concrete instantiations of identification protocols are often defined as three-move challenge-response protocols, in which P first sends a *commitment* $t \leftarrow P_t(\mathsf{state}_P)$ to V, V then sends a *challenge* $c \leftarrow V_c(t, \mathsf{state}_V)$ back to P, and finally P sends the *response* $s \leftarrow P_s(c, \mathsf{state}_P)$ to V. At the end of this interaction, V executes $v \leftarrow V_v(s, \mathsf{state}_V)$ to obtain the protocol output $v \in \{\mathsf{accept}, \mathsf{reject}\}$. Such a construction $\mathcal{I} = (G, P, V)$ with two pairs of algorithms $P = (P_t, P_s)$ and $V = (V_c, V_v)$ is sometimes called *canonical identification protocol* [6]. In this particular case, the generated conversation transcripts are triples (t, c, s). A transcript that leads to $v = \mathsf{accept}$ is called *accepting conversation* for vk.

2.2 Security Properties

The security of an identification protocol can be defined in terms of *direct*, *passive*, or *active* adversaries, with direct adveraries being the least powerful and active adversaries the most powerful ones. In corresponding *impersonation attack games*, a *challenger* executes the key generation algorithm $(sk, vk) \leftarrow G()$ and sends vk to an adversary A. The adversary's goal is to interact with the challenger following the verifier's algorithm V (with input vk) in a way, such that at the end of the protocol, V outputs accept. This is called an *impersonation attempt*, and we say that A wins the attack game, if the impersonation attempt has been successful.

While vk is the only input for direct adversaries to win the impersonation attack game, passive adversaries are allowed to run an *eavesdropping phase*, in which they can request transcripts of conversations between P and V from the challenger. To comply with this request, the challenger runs the interaction between P and V multiple times, each time with P initialized with sk and V initialized with vk. All generated transcripts are sent to A, who can use them as additional inputs to the impersonation attempt.

In addition to running an eavesdropping phase, active adversaries have a further option of running an *active probing phase*, in which multiple instances of the protocol are executed with the challenger playing the role of the prover following algorithm P with input sk, and the adversary playing the role of the verifier, not necessarily following algorithm V. Whatever the adversary learns from the challenger can be used as additional inputs to the impersonation attempt.

By calling these attack games ID1 (direct attack), ID2 (passive attack), and ID3 (active attack), we can define corresponding probabilities of A winning the attack game as $\mathrm{ID1}\mathsf{adv}[A, \mathcal{I}]$, $\mathrm{ID2}\mathsf{adv}[A, \mathcal{I}]$, and $\mathrm{ID3}\mathsf{adv}[A, \mathcal{I}]$, respectively.[4] An identification protocol $\mathcal{I} = (G, P, V)$ is *secure against direct attacks*, if $\mathrm{ID1}\mathsf{adv}[A, \mathcal{I}]$ is negligible for all adversaries A. Similarly, $\mathcal{I}$ is called *secure against passive attacks*, if $\mathrm{ID2}\mathsf{adv}[A, \mathcal{I}]$ is negligible, and *secure against active attacks*, if $\mathrm{ID3}\mathsf{adv}[A, \mathcal{I}]$ is negligible.

An identification protocol $\mathcal{I} = (G, P, V)$ is called *honest-verifier zero-knowledge* (HVZK), if there exists an efficient probabilistic algorithm Sim, called *simulator*, which has an output distribution on input vk that is identical as for the transcripts obtained from a conversation between $P(sk)$ and $V(vk)$, for all possible key pairs $(sk, vk) \leftarrow G()$. This notion is useful for proving that $\mathcal{I}$ is secure against passive attacks, because according to Theorem 19.3 in [1], security against direct attacks and HVZK implies security against passive attacks. The general strategy of our security proof in Section 4 makes use of this theorem.

2.3 Sigma-Protocols

In our security proof in Section 4, proving that our protocol is secure against direct attacks will be based on the notion of *Sigma-protocols*. Clearly, canonical

[4] This notation is adopted from [1], where adv stands for the adversary's advantage.

identification protocols as defined in Section 2.1 have the structure of a Sigma-protocol, but Sigma-protocols are slightly more general. They consist of a pair of algorithms $\Sigma = (P, V)$, where again P models the prover and V the verifier. For a binary relation $R \subseteq X \times Y$, the value x of $(x, y) \in R$ is called *witness* of the corresponding *statement* y. The intuitive goal of executing the protocol is to convince V (on input y) that the witness x of the statement y is known to P (on input x, y).

As in the case of canonical identification protocols, both the prover and the verifier consist of a pair of algorithms $P = (P_t, P_s)$ and $V = (V_c, V_v)$, respectively, which they apply in the usual order. Conversation transcripts of a protocol execution are therefore triples (t, c, s), but with the particularity that the challenge c is selected uniformly at random from a sufficiently large *challenge space* $\mathcal{C}$ (with $|\mathcal{C}|^{-1}$ being negligible). An honest verifier strictly sticks to this rule. In a Sigma-protocol for R, after interacting with $P(x, y)$, $V(y)$ outputs accept with probability 1 for every $(x, y) \in R$.

A protocol $\Sigma = (P, V)$ for $R \subseteq X \times Y$ provides *knowledge soundness*, if there is an efficient deterministic algorithm Ext, called *witness extractor*, which for two accepting protocol transcripts (t, c, s) and (t, c', s'), $c \neq c'$, always outputs $x \in X$ such that $(x, y) \in R$, i.e., such that $x \leftarrow \mathsf{Ext}(y, t, c, s, c', s')$ is a witness for y. In our security proof in Section 4, we will have to demonstrate that our protocol offers knowledge soundness.

Every protocol $\Sigma = (P, V)$ with knowledge soundness can be transformed into an identification protocol, provided that an efficient key generation algorithm G that generates keys $sk = (x, y)$ and $vk = y$ for some $(x, y) \in R$ can be defined. For the security of the resulting identification protocol $\mathcal{I} = (G, P, V)$, it should be hard for a given $y \in Y$ generated by G to compute $\hat{x} \in X$ such that $(\hat{x}, y) \in R$. Theorem 19.14 in [1] states that $\mathcal{I}$ constructed from Σ is secure against direct attacks, under the conditions that Σ provides knowledge soundness and that G is "one-way" in the sense just discussed above (we refer to [1] for a more formal definition of this property).

In the same way as for identification protocols, it is possible to define the HVZK property for Sigma-protocols. However, it is common and sometimes more convenient to define a related property called *special HVZK* (which implies HVZK). We will also use this notion in our proof of Section 4 to demonstrate that our protocol is secure against passive attacks (using Theorem 19.15 from [1]). If $\Sigma = (P, V)$ is a Sigma-protocol for $R \subseteq X \times Y$, special HVZK means that there exists an efficient probabilistic algorithm Sim, called *simulator*, that takes $(y, c) \in Y \times \mathcal{C}$ as input and always computes a pair (t, s) such that (t, c, s) is an accepting conversation for y, with an output distribution of Sim that is identical to the distribution obtained from a real conversation between $P(x, y)$ and $V(y)$.

2.4 From Identification Protocols to Signature Schemes

Just as Sigma-protocols can be transformed into secure identification protocols, it is possible to convert them into secure signature schemes with message space $\mathcal{M}$. The general idea of this transformation, which is originally due to Fiat and Shamir [3], consists in computing the challenge c using a hash function $H : \mathcal{M} \times \mathcal{T} \rightarrow \mathcal{C}$, where $\mathcal{T}$ denotes the set of all possible commitments t obtained from P_t. The resulting *Fiat-Shamir signature scheme* consists of the following algorithms:

- A one-way key generation algorithm G for $R \subseteq X \times Y$, which outputs a private signature key $sk = (x, y) \in R$ and a public verification key $vk = y \in Y$.
- A signing algorithm $\mathsf{Sign}(m, sk)$ with output $\sigma = (t, s)$, which uses P_t to compute t, H to compute $c \leftarrow H(m, t)$, and P_s to compute s.
- A verification algorithm $\mathsf{Verify}(m, \sigma, vk)$ with output $v \in \{\mathsf{accept}, \mathsf{reject}\}$, which uses H to compute $c \leftarrow H(m, t)$ and V_v to compute v.

According to Theorem 19.16 in [1], $\mathcal{S} = (G, \mathsf{Sign}, \mathsf{Verify})$ is a secure signature scheme, if $\mathcal{I} = (G, P, V)$ is secure against passive attacks and if H is modeled as a random oracle.[5] Note that Sign and Verify can be implemented as stateless algorithms.

2.5 Application to E-Voting

As discussed in the previous subsection, it is always possible to derive a secure signature scheme from a passively secure identification protocol using the Fiat-Shamir transformation. This is of particular interest to our e-voting use case for signing the ballots submitted to the public bulletin board. In this way, everyone can execute the Verify algorithm on the attached signatures to verify the eligibility of the voters (under the condition that everyone agrees on the list of public keys of all eligible voters).

If we use the above construction to implement eligibility verifiability based on a HVZK identification protocol, secret keys remain completely hidden when they are used to cast a vote. This is an appealing property, because it allows reusing the keys arbitrarily many times without giving the adversary any advantage. Clearly, in order to perform the necessary cryptographic computations with sk as input, the voter depends on a trustworthy machine for doing so. However, like in the Swiss e-voting use case, it is often assumed that voting clients are not trustworthy (at least not for vote integrity). Nevertheless, we think that the HVZK property is still useful, for example to allow the recovery of a lost vote casting session.

Another useful property in the e-voting use case is *key aggregation*, which means that multiple key pairs (sk_i, vk_i) can be merged into a single key pair (sk, vk). This can be used to generate the keys in a distributed manner by a group of parties of which only one needs to be trusted. A precondition for key aggregation in a setting with keys taken from $R \subseteq X \times Y$ is the existence of a binary operation $\otimes : R \times R \rightarrow R$ that can be applied for merging $(x_1, y_1) \in R$ and $(x_2, y_2) \in R$ into $(x_1, y_1) \otimes (x_2, y_2) \in R$. In many cases, this operation results from applying given group operations on X and Y component-wise to the two inputs. In our protocol presented in the next section, this will be the case.

[5] An additional technical property called *unpredictable commitments* must be satisfied for $\Sigma = (P, V)$, but this is almost always the case in an obvious way [1].

3 Pairing-Based Identification With Short Keys

In this section, we present our new identification protocol with secret keys as short as the security parameter σ. As our protocol requires a setting based on pairings, we first give a short introduction to pairing-based cryptography. We assume that readers are sufficiently familiar with concepts such as groups or elliptic curves.

3.1 Pairing-Based Cryptography

In cryptography, a *pairing* is a bilinear map $\theta :\rightarrow G_1 \times G_2 \rightarrow G_T$ between cyclic groups G_1, G_2, and G_T, all of prime order q, which satisfies the following properties for all $x \in G_1$ and $y \in G_2$:

- *Bilinearity*: $\theta(x^a, y^b) = \theta(x, y)^{ab}$, for all $a, b \in \mathbb{Z}$.
- *Non-degeneracy*: if $x \neq 1$ and $y \neq 1$, then $e(x, y) \neq 1$.
- *Efficiency*: θ is efficiently computable.

The special case of $G = G_1 = G_2$ is called *Type-1 pairing*. For $G_1 \neq G_2$, θ is called *Type-2 pairing*, if there exists an efficiently computable homomorphism $\Psi : G_2 \leftarrow G_1$, and *Type-3 pairing*, if no such homomorphism exists in either direction [4]. The *Tate* and the *Weil* pairings are the most efficient ones and hence most relevant for practical applications [8]. In both cases, G_1 and G_2 are q-order subgroups of elliptic curves over finite fields such as $E(\mathbb{F}_p)$ or $E(\mathbb{F}_{p^k})$, while G_T is a q-order subgroup of $\mathbb{F}_{p^k}$ (k denotes the curve's embedding degree). For cryptographic applications, it is important to assume that it is difficult to invert θ or to solve the discrete logarithm problem in G_1, G_2, or G_T.

For the purpose of this paper, we are free to choose a symmetric (Type-1) or an asymmetric (Type-2 or Type-3) pairing. There is a Type-3 setting with efficient group operations in G_1 and an efficient pairing θ for a wide flexibility of parameters, but with k^2-less-efficient group operations in G_2. We might also opt for a Type-1 setting with efficient group operations in G, but with less flexible parameters and the performance relative to θ degrading faster with higher security levels [4]. As a general rule, we require the group size to consist of $||q|| \geq 2\sigma$ bits, where σ denotes the security parameter.

3.2 Protocol Description

Our new identification protocol has the structure of a canonical protocol $\mathcal{I} = (G, (P_t, P_s), (V_c, V_v))$ with two pairs of algorithms for each party. These algorithms operate on elements of the q-order groups G_1, G_2, and G_T of a publicly known pairing θ, and with exponents taken $\mathbb{Z}_q$. As discussed earlier, we require q to be at least twice as big as the security parameter σ. For maximum convenience, we simply assume $||q|| = 2\sigma$. Furthermore, we assume that independent generators $g_1 \in G_1 \setminus \{1\}$ and $g_2 \in G_2 \setminus \{1\}$ have been selects in a verifiable matter and are known to both the prover and the verifier.

Key Generation. The general idea of the key generation is to select two random values $r \in \mathbb{Z}_q$ and $x \in \mathbb{Z}_{2^\sigma}$ of different sizes. We get security against DL-breaking algorithms due to the large randomness r, ~~while at the same time keeping the secret key $sk = x$ as short as the security parameter permits~~. As shown in Algorithm 1, we use r and x to compute a pair of elements $y = (y_1, y_2) = (g_1^r, g_2^{r+x}) \in G_1 \times G_2$, which forms the verification key $vk = y$. Since x is used as an exponent in computations over G_2, we consider it an element of $\mathbb{Z}_q$.

Algorithm 1 $G()$

```
1: $(r, x) \xleftarrow{\$} \mathbb{Z}_q \times \mathbb{Z}_{2^\sigma}$
2: $(y_1, y_2) \leftarrow (g_1^r, g_2^{r+x})$
3: $y \leftarrow (y_1, y_2)$
4: return $(sk, vk) = (x, y)$        // $(sk, vk) \in \mathbb{Z}_q \times (G_1 \times G_2)$
```

But: e(1/y, g1)*e(g2, x) = e(g1, g2)^x
This exposes a known group element
raised to the short secret, hence we
again need to double the short secret
size for the appropriate security
parameter

Prover. The general idea of the prover's algorithms is to generate a second verification key $\hat{y} = (\hat{y}_1, \hat{y}_2)$ for the same secret key x, but using a different randomization $\hat{r}$. To avoid trivial constructions of $\hat{y}$ by an adversary not knowing x, for example by simply replaying the verification key y, P needs to prove knowledge of the fresh randomness $\hat{r}$ and of $\hat{r} + x$. We realize this proof of knowledge using standard zero-knowledge techniques. Algorithms 2 and 3 describe respective computations for obtaining a commitment t and for generating the response s to the verifier's challenge c. Note that since both P_t and P_s are interactive protocol algorithms, they update the prover's state state_P.

Algorithm 2 $P_t(\mathsf{state}_P)$

```
1: select $sk = x$ from $\mathsf{state}_P$
2: $\hat{r} \xleftarrow{\$} \mathbb{Z}_q$
3: $(\hat{y}_1, \hat{y}_2) \leftarrow (g_1^{\hat{r}}, g_2^{\hat{r}+x})$
4: $(r_1, r_2) \xleftarrow{\$} \mathbb{Z}_q \times \mathbb{Z}_q$
5: $(t_1, t_2) \leftarrow (g_1^{r_1}, g_2^{r_2})$
6: $t \leftarrow (\hat{y}_1, \hat{y}_2, t_1, t_2)$
7: add $\hat{r}, r_1, r_2, t$ to $\mathsf{state}_P$
8: return $t$        // $t \in G_1 \times G_2 \times G_1 \times G_2$
```

Algorithm 3 $P_s(c, \mathsf{state}_P)$

```
1: select $sk = x, \hat{r}, r_1, r_2$ from $\mathsf{state}_P$
2: $(s_1, s_2) \leftarrow (r_1 + c\,\hat{r}, r_2 + c\,(\hat{r} + x))$
3: $s \leftarrow (s_1, s_2)$
4: add $c, s$ to $\mathsf{state}_P$
5: return $s$        // $s \in \mathbb{Z}_q \times \mathbb{Z}_q$
```

Verifier. The challenge selected by the verifier in Algorithm 4 is a random integer from $\mathbb{Z}_{2^\sigma}$. Upon receiving the prover's response in Algorithm 5, V first checks if $y = (y_1, y_2)$ and $\hat{y} = (\hat{y}_1, \hat{y}_2)$ are both valid verification keys for the same secret key x. V can perform this check without knowing x by applying the pairing twice and testing the two results for equality. Afterwards, V needs to check the prover's claim of knowing both $\hat{r}$ and of $\hat{r} + x$. If all checks succeed, V accepts the prover's identity claim. Again, note that both algorithm V_c and V_v update the verifier's state state_V.

Algorithm 4 $V_c(t, \mathsf{state}_V)$

```
1: $c \xleftarrow{\$} \mathbb{Z}_{2^\sigma}$
2: add $t, c$ to $\mathsf{state}_V$
3: return $c$        // $c \in \mathbb{Z}_{2^\sigma}$
```

Algorithm 5 $V_v(s = (s_1, s_2), \mathsf{state}_V)$

```
1: $v \leftarrow \mathsf{accept}$
2: select $vk = (y_1, y_2), t = (\hat{y}_1, \hat{y}_2, t_1, t_2), c$ from $\mathsf{state}_V$
3: if $\theta(y_1 \times (\hat{y}_1)^{-1}, g_2) \neq \theta(g_1, y_2 \times (\hat{y}_2)^{-1})$ then
4:     $v \leftarrow \mathsf{reject}$
5: else if $g_1^{s_1} \neq t_1 \times (\hat{y}_1)^c$ then
6:     $v \leftarrow \mathsf{reject}$
7: else if $g_2^{s_2} \neq t_2 \times (\hat{y}_2)^c$ then
8:     $v \leftarrow \mathsf{reject}$
9: add $s, v$ to $\mathsf{state}_V$
10: return $v$        // $v \in \{\mathsf{accept}, \mathsf{reject}\}$
```

3.3 Discussion

To show that $\mathcal{I} = (G, (P_t, P_s), (V_c, V_v))$ is an identification protocol, we must check that the inequalities in Lines 3, 5, and 7 of Algorithm 5 are false for all $(sk, vk) \leftarrow G()$, or equivalently, that corresponding equalities are always true. This is rather obvious in all three cases:

$$\begin{aligned}
&\theta(y_1 \times (\hat{y}_1)^{-1}, g_2) = \theta(g_1, y_2 \times (\hat{y}_2)^{-1}) \\
&\iff \theta(g_1^r \times (g_1^{\hat{r}})^{-1}, g_2) = \theta(g_1, g_2^{r+x} \times (g_2^{\hat{r}+x})^{-1}) \\
&\iff \theta(g_1^{r-\hat{r}}, g_2) = \theta(g_1, g_2^{r-\hat{r}}) \iff \theta(g_1, g_2)^{r-\hat{r}} = \theta(g_1, g_2)^{r-\hat{r}}, \qquad \square
\end{aligned}$$

$$g_1^{s_1} = t_1 \times (\hat{y}_1)^c \iff g_1^{r_1 + c\,\hat{r}} = g_1^{r_1} \times (g_1^{\hat{r}})^c \iff g_1^{r_1 + c\,\hat{r}} = g_1^{r_1 + c\,\hat{r}}, \qquad \square$$

$$g_2^{s_2} = t_2 \times (\hat{y}_2)^c \iff g_2^{r_2 + c\,(\hat{r}+x)} = g_2^{r_2} \times (g_2^{(\hat{r}+x)})^c \iff g_2^{r_2 + c\,(\hat{r}+x)} = g_2^{r_2 + c\,(\hat{r}+x)}. \qquad \square$$

The performance of the proposed protocol mainly depends on the number of group exponentiations and pairings necessary for executing the protocol. The prover requires two exponentiations for generating a key pair (one in G_1 and one in G_2) and four exponentiations for executing the protocol honestly (two in G_1 and two in G_2). The honest verifier also requires four exponentiations for executing the protocol, but additionally uses the pairing θ twice. Therefore, given the usual high computational costs for computing pairings, we expect the verifier to be considerably less efficient than the prover.

Key generation as defined in Algorithm 1 offers a natural way of merging two key pairs $(x_1, (y_{1,1}, y_{1,2})) \in \mathbb{Z}_q \times (G_1 \times G_2)$ and $(x_2, (y_{2,1}, y_{2,2})) \in \mathbb{Z}_q \times (G_1 \times G_2)$ into a combined key pair,

$$\begin{aligned}
(x, (y_1, y_2)) &= (x_1, (y_{1,1}, y_{1,2})) \otimes (x_2, (y_{2,1}, y_{2,2})) \\
&= (x_1 + x_2, (y_{1,1} \times y_{2,1}, y_{2,1} \times y_{2,2})) \in \mathbb{Z}_q \times (G_1 \times G_2),
\end{aligned}$$

by applying the group operation component-wise, i.e., $+$ denotes modular addition modulo q and $\times$ respective group operations in G_1 and G_2. Note that if both

x_1 and x_2 are of length σ as defined in Algorithm 1, then $x = x_1 + x_2 \in \mathbb{Z}_q$ is of length $\sigma + 1$, i.e., key aggregation slightly increases the length of the secret key. Generally, merging n equally long keys increases the length of the resulting key by up to $\log_2 n$ bits. For the intended application of this paper, this seems acceptable.

Regarding the choice of pairing, the protocol as defined in Algorithms 1 to 5 is designed for the most general case of an asymmetric Type-3 pairing, which includes Type-2 pairings and symmetric Type-1 pairings as special cases. We are therefore free to choose any suitable paring. Note that the assumption that g_1 and g_2 are independent gets important in the symmetric case $G = G_1 = G_2$, because otherwise an adversary in possession of $z = \log_{g_2} g_1$ can derive the secret key $x = \log_{g_2}(y_2 \times y_1^{-z})$ from the verification key $y = (y_1, y_2)$ in $O(\sqrt{2^\sigma})$ time.

4 Security Analysis

In this section, we prove that the identification protocol $\mathcal{I} = (G, P, V)$ from Section 3 is secure against passive attack. The general proof strategy is based on Theorems 9.14 and 9.15 from [1]. Recall from Section 2.3 that if $\Sigma = (P, V)$ is a Sigma-protocol with knowledge soundness and if G is one-way, then $\mathcal{I} = (G, P, V)$ is secure against direct attacks (Theorem 9.14). Furthermore, if Σ is special HVZK, then $\mathcal{I}$ is secure against passive attacks (Theorem 9.15). This general strategy implies the following proof steps:

1. Show that $\Sigma = (P, V)$ is a Sigma-protocol.
2. Show that $\Sigma = (P, V)$ provides knowledge soundness.
3. Show that G is one-way.
4. Show that $\Sigma = (P, V)$ is special HVZK.[6]

Following the procedure described in Section 2.4, we can then use our passively secure identification protocol $\mathcal{I} = (G, P, V)$ to construct a secure Fiat-Shamir signature scheme $\mathcal{S} = (G, \mathsf{Sign}, \mathsf{Verify})$. In the random oracle model, the security of this scheme is guaranteed by Theorem 9.15 from [1]. This is the scheme we propose to use in our e-voting use case to obtain considerably shorter keys.

Theorem 1. *$\Sigma = (P, V)$ is a Sigma-protocol.*

Proof. Since $\mathcal{I} = (G, P, V)$ is canonical with two pairs of algorithms $P = (P_t, P_s)$ and $V = (V_c, V_s)$, it already has the general structure of a Sigma-protocol. In particular, V_c outputs a random challenge c from $\mathcal{C} = \mathbb{Z}_{2^\sigma}$. Clearly, $|\mathcal{C}|^{-1} = 2^{-\sigma}$ is negligible in σ. The relation $R \subseteq X \times Y$ follows from G, which generates key

[6] While special HVZK is not required for the proof construction used in this paper, it is a requirement for other proof constructions such as the OR-proof construction in [1].

pairs $(x, y) \in \mathbb{Z}_q \times (G_1 \times G_2)$.[7] This implies $X = \mathbb{Z}_q$, $Y = G_1 \times G_2$, and hence

$$R = \{(x, y) : x \in \mathbb{Z}_q \wedge \exists r \in \mathbb{Z}_q \text{ s.t. } y = (g_1^r, g_2^{r+x})\}.^8$$

In Section 3.3, we have already shown that the verifier outputs accept for all possible key pairs $(x, y) \leftarrow G()$ and therefore for all elements of R. Hence, all properties of a Sigma-protocol a given. □

Theorem 2. *$\Sigma = (P, V)$ provides knowledge soundness.*

Proof. Let (t, c, s) and (t, c', s') be two different accepting conversations for $(x, (y_1, y_2)) \in R$ for the same t. We can define an extractor that outputs $x \leftarrow \mathsf{Ext}(y, t, c, s, c', s')$ as follows:

1. Compute

$$R_1 := \frac{s_1 - s_1'}{c - c'} = \frac{r_1 + c\,\hat{r} - r_1 - c'\,\hat{r}}{c - c'} = \frac{(c - c')\,\hat{r}}{c - c'} = \hat{r},$$
$$R_2 := \frac{s_2 - s_2'}{c - c'} = \frac{r_2 + c(\hat{r} + x) - r_2 - c'(\hat{r} + x)}{c - c'} = \frac{(c - c')(\hat{r} + x)}{c - c'} = \hat{r} + x.$$

2. Return $x = R_2 - R_1$. □

Theorem 3. *G is one-way under the DL assumption for G_1 and G_2.*

Proof. We have to prove that computing the unique[8] $x \in X$ such that $(x, y) \in R$ is hard for a given $y = (y_1, y_2) \in Y$. Clearly, one can find x by solving DL in G_1 to obtain r satisfying $y_1 = g^r$ and solving DL in G_2 to obtain $r + x$ satisfying $y_x = g^{r+x}$. The hardness of DL in both G_1 and G_2 is therefore necessary for a necessary for making G one-way. In the case of $G_1 = G_2$, assuming that g_1 and g_2 are independent prevents reducing (y_1, y_2) to either g_1^x or g_2^x, from which x can be found in $O(\sqrt{2^\sigma})$ time (see discussion in Section 3.3).

Note that this is not a proof as it only argues about necessary conditions, but not sufficient.

Corollary 1. *$\mathcal{I} = (G, P, V)$ is secure against direct adversaries under the DL assumption for G_1 and G_2.*

Proof. We know from Theorems 1 and 2 that $\Sigma = (P, V)$ is a Sigma-protocol with knowledge soundness and from Theorem 3 that G is one-way. Hence, all preconditions of Theorem 9.14 in [1] are met and our claim follows.

Theorem 4. *$\Sigma = (P, V)$ is special HVZK.*

Proof. To show that $\Sigma = (P, V)$ is special HVZK, we need to simulate accepting conversations without knowing the secret x, such that the resulting transcripts are indistinguishable from real transcripts. In our protocol, the distribution of the values $\hat{y}_1$, $\hat{y}_2$, t_1, t_2, c, s_1, s_2 are as follows:

[7] For proving that Σ is a Sigma-protocol, it is not important that the bit length of x is restricted to σ. Without loss of generality, we can therefore assume that x is an element of $\mathbb{Z}_q$.

[8] Note that in R, every statement $y \in G_1 \times G_2$ has a unique witness x, but every $x \in \mathbb{Z}_q$ is a witness for q different statements y (depending on the choice of r).

- $\hat{y}_1$ is obtained from raising g_1 to the power of a uniform random value $\hat{r} \in \mathbb{Z}_q$.
- $\hat{y}_2$ is obtained from raising g_2 to the power of a uniform random value $\hat{r} + x \in \mathbb{Z}_q$.
- t_1 is obtained from raising g_1 to the power of a uniform random value $r_1 \in \mathbb{Z}_q$. The same holds for t_2 relative to g_2 and $r_2 \in \mathbb{Z}_q$.
- s_1 is obtained from adding the uniform random value r_1 to another value. [9]. The same holds for s_2 relative r_2.

We summarize that all values except $\hat{y}_2$ are distributed uniformly at random. We avoid describing the distribution of $\hat{y}_2$ exactly, but rather observe that the distribution of $\hat{y}_2$ relates to the distribution of $\hat{y}_1$ in the same way as the distribution of y_2 relates to the distribution of y_1 in the given verification key. This is a consequence of the fact that (y_1, y_2) is generated by $G()$ in the same way as the $(\hat{y}_1, \hat{y}_2)$ is generated by P_t.

To simulate an accepting conversation without knowing x, replicating the observed distributions of a real transcript given the challenge $c \in \mathcal{C}$, we propose to choose the values of $((\hat{y}'_1, \hat{y}'_2, t'_1, t'_2), c, (s'_1, s'_2)) \leftarrow \mathsf{Sim}(y, c)$ of the generated transcript as follows:

$$\begin{aligned}
\hat{r}', s'_1, s'_2 &\xleftarrow{\$} \mathbb{Z}_q, \\
(\hat{y}'_1, \hat{y}'_2) &\leftarrow (y_1 \times g_1^{\hat{r}'}, y_2 \times g_2^{\hat{r}'}), \\
(t'_1, t'_2) &\leftarrow (g_1^{s'_1} \times (\hat{y}'_1)^{-c}, g_2^{s'_2} \times (\hat{y}'_2)^{-c}).
\end{aligned}$$

We argue that indeed the distribution of the obtained values is identical to a real protocol run:

- c, s'_1 and s'_2 are distributed uniformly at random.
- $\hat{y}'_1$ is distributed uniformly at random: As $\hat{r}'$ is chosen uniformly at random, using it as an exponent for g_1, results in a uniform random value. Multiplying it to $\hat{y}_1$ "shifts" $\hat{y}_1$ by the uniform random amount, resulting in a uniform at random distribution.
- $\hat{y}'_2$ replicates the specific distribution relative to $\hat{x}'_1$ as observed in a real protocol run: We use the same uniform random $\hat{r}'$ as an exponent for g_2 and multiply this to $\hat{y}_2$, hence preserving the relative distribution between $\hat{y}_2$ and $\hat{y}_1$ for $\hat{y}'_1$ and $\hat{y}'_2$.
- t'_1 is distributed uniformly at random: As s'_1 is chosen uniformly at random, using it as an exponent for g_1 results in a uniform random value. The second term $(\hat{y}'_1)^{-c}$ is also a uniform random value following the same argumentation. Multiplying two uniform random values results in a uniform distribution. The same argumentation applies to t'_2 relative to g_2, s'_2, and $\hat{y}'_2$.

As a final step, we need to ensure the generated transcript also passes the verifiers's correctness checks. As in Section 3.3, there are three checks to perform,

[9] Note that r_1 was already used to calculate t_1, but if DL is assumed to be hard in G_1, this gives no advantage to an adversary.

one involving the pairing and two for checking the relationship between t'_1, c and s'_1 (respectively t_2, c, s_2):

$$\begin{aligned}
&\theta(y_1 \times (\hat{y}'_1)^{-1}, g_2) = \theta(g_1, y_2 \times (\hat{y}'_2)^{-1}) \\
&\iff \theta(y_1 \times (y_1 \times g_1^{\hat{r}'})^{-1}, g_2) = \theta(g_1, y_2 \times (y_2 \times g_2^{\hat{r}'})^{-1}) \\
&\iff \theta(g_1^{-\hat{r}'}, g_2) = \theta(g_1, g_2^{-\hat{r}'}) \iff \theta(g_1, g_2)^{-\hat{r}'} = \theta(g_1, g_2)^{-\hat{r}'}, \qquad \square
\end{aligned}$$

$$g_1^{s'_1} = t'_1 \times (\hat{y}_1)^c \iff g_1^{s'_1} = g_1^{s'_1} \times (\hat{y}'_1)^{-c} \times (\hat{y}_1)^c \iff g_1^{s'_1} = g_1^{s'_1}, \qquad \square$$

$$g_2^{s'_2} = t'_2 \times (\hat{y}_2)^c \iff g_2^{s'_2} = g_2^{s'_2} \times (\hat{y}'_2)^{-c} \times (\hat{y}_2)^c \iff g_2^{s'_2} = g_2^{s'_2}. \qquad \square$$

Corollary 2. *$\mathcal{I} = (G, P, V)$ is secure against passive adversaries under the DL assumption for G_1 and G_2.*

Proof. We know from Corollary 1 that $\Sigma = (P, V)$ is secure against direct attacks and from Theorem 4 that Σ is special HVZK. Hence, all preconditions of Theorem 9.15 in [1] are met and our claim follows.

5 Conclusion

The pairing-based identification protocol presented in this paper has a number of useful properties. It is provably secure against passive attacks based on the DL assumption, ~~it generates secret keys that are no longer than the security parameter,~~ it supports key aggregation without making them considerably larger, and it is sufficiently efficient for practical applications. We see it therefore as an alternative to existing identification methods in applications such as e-voting, particularly in a context where keys are generated in distributed manner and given to the voters in printed form. The approaches currently developed in Switzerland could become potential beneficiaries of this method.

References

1. Boneh, D., Shoup, V.: A graduate course in applied cryptography – version 0.5. Available at cryptobook.us (2020)
2. Esseiva, O.: Protocol of the Swiss Post voting system – Version 0.9.10. Tech. rep., Swiss Post Ltd., Bern, Switzerland (2021)
3. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86, 6th Annual International Cryptology Conference on Advances in Cryptology. pp. 186–194. LNCS 263, Santa Barbara, USA (1986)
4. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**(16), 3113–3121 (2008)

5. Haenni, R., Koenig, R.E., Locher, P., Dubuis, E.: CHVote protocol specification – version 3.2. IACR Cryptology ePrint Archive **2017/325** (2020)
6. Katz, J.: Digital Signatures. Springer (2010)
7. Kremer, S., Ryan, M., Smyth, B.: Election verifiability in electronic voting protocols. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS'10, 5th European Conference on Research in Computer Security. pp. 389–404. Athens, Greece (2010)
8. Lynn, B.: On The Implementation Of Pairing-Based Cryptosystems. Ph.D. thesis, Stanford University (2007)
9. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Brickell, E.F. (ed.) CRYPTO'92, 12th Annual International Cryptology Conference on Advances in Cryptology. pp. 31–53. LNCS 740, Santa Barbara, USA (1992)
10. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4**(3), 161–174 (1991)
11. Smyth, B., Frink, S., Clarkson, M.R.: Election verifiability: Cryptographic definitions and an analysis of Helios, Helios-C, and JCJ. IACR Cryptology ePrint Archive **2015/233** (2015)