Evaluate a Pairing-Based Identification Protocol

Florian Moser

March 19, 2021

Abstract

CHVote is an E-Voting protocol supporting direct democracy in Switzerland. As part of the voting procedures, the voters have to authenticate their vote. To improve usability of authentication, a pairing-based identification protocol is proposed which allows shorter voter-held secrets under the same security level than the currently pursued approach. We describe the proposed protocol, and voter authentication in general in *CHVote* . We proof a pairing-based identification protocol with shorter secrets to be secure against impersonation both under active as well as passive attacks, and derive a EUF-CMA secure signature scheme. Finally, we propose possible alternatives for future research which also shorten voter-held secrets.

Contents

1 Introduction **2**

2 Cryptographic Primitives **3**
- 2.1 Cryptographic basics . . . 4
- 2.2 Proofs of Knowledge . . . 4
- 2.3 Sigma Protocols and Properties . . . 6
- 2.4 Identification protocols . . . 8
- 2.5 Signature schemes . . . 9

3 Authentication in *CHVote* **10**
- 3.1 Authentication within *CHVote* . . . 10
- 3.2 Authenticating messages . . . 11
- 3.3 Proposed non-interactive identification protocol . . . 11

4 A pairing-based identification protocol **14**
- 4.1 Sigma protocol . . . 14
- 4.2 IMP-AA identification protocol . . . 14
- 4.3 IMP-PA identification protocol . . . 16
- 4.4 EUF-CMA signature scheme . . . 18
- 4.5 Pairing implementation notes . . . 20

5 Alternatives **22**

1 Introduction

The *CHVote* specification describes an E-Voting protocol supporting direct democracy as implemented and practiced in Switzerland. It provides End-to-End Encryption, Individual and Universal Verifiability and Distribution of trust. We refer for a detailed discussion of these properties to the specification [HKLD17]. *CHVote* is developed at the University of Applied Sciences of Bern since 2016 (up until 2018 for the State of Geneva), and has been updated in regular intervals. This report is based on Version 3.1 (released at 01.10.2020). An earlier version of this specification has been formally analyzed [BCG+18].

As part of the voting procedures, the voters have to authenticate their vote. In the proposed implementation, each voter receives two secrets over a secure channel. During the voting procedure, they transfer (e.g. by typing) this secret into the voting client to create and confirm their vote. Clearly, a shorter secret improves the usability of this part of the voting procedure (e.g. less to type).

The security of the currently by *CHVote* used authentication scheme depends on the size of the mathematical group it operates in. The underlying hardness assumption requires this size to be double the security parameter, to offset the root in runtime of the fastest currently known hardness-breaking algorithms. As the secret is chosen from this same group, its length is double the security parameter, too.

The *Protocol* [1] allows to use shorter secrets, as the length of the secret is no longer bound by group the scheme operates in. This way, the group be chosen suitably large to be secure against the fastest currently known hardness-breaking algorithms, while the length of the secret can be chosen just as large as the security parameter.

Contributions We give a concise summary of how authentication in *CHVote* works, and describe how the *Protocol* works. Based on its ideas, we derive an identification scheme which we proof *secure against impersonation under active attacks* and *secure against impersonation under passive attacks*. We further apply the Fiat-Shamir construction to derive a signature scheme that is existential-unforgeable under chosen-message attacks. Finally, given the fundamental requirement of a small secret size, we propose two alternatives likely suitable for *CHVote* for further research: Using a pseudo-random generator or using a modified version of Schnorr's Signature scheme.

[1] for Version 3.1 of the *CHVote* specification, described in section *11.2.1 Approach 1: Using Bilinear Mappings*

2 Cryptographic Primitives

We clarify notation, terminology and cryptographic primitives we use throughout this work.

Notation We use upper-case letters for sets and lower-case letters for their elements (like $X = \{x_1, x_2, ..., x_n\}$). $|X|$ denotes the cardinality of the set X.

For algorithms, we use bold letters (like **Sign**). We denote $\overset{\$}{\leftarrow}$ when we choose an element uniform at random (like $x \overset{\$}{\leftarrow} X$). We forgo modeling the source of randomness explicitly; we simply assume each algorithm has access to randomness of suitable size.

Terminology We write *secure channel* when we assume a message transferred between two parties is not altered or read by anyone. *Broadcast* means the same message is sent from a single sender to multiple receivers.

A *public key* can be publicly known, and allows to verify a signature (or to encrypt a message). The corresponding *private key* must only be known to the signer (or decryptor).

Groups A (multiplicative) group $\Gamma = (G, *, ^{-1}, 1)$ is an algebraic structure consisting of a set G of elements, the binary operation $* : G \times G \to G$, the unary operation $^{-1} : G \to G$, and the neutral element 1. For any $\{x, y, z\} \subset G$, the following holds: Associativity ($(x * y) * z = x * (y * z)$), identity element ($1 * x = x * 1 = x$), and inverse element ($x * x^{-1} = 1$). x^k applies the group operator $k - 1$ times to x. An element $g \in G$ is called a generator of G, iff $\{g^1, ..., g^p\} = G$ for $p = |G|$. If the group is of prime order, every element is a generator, except the neutral element 1. After this paragraph, our notation will refer to Γ using G.

With $\mathbb{Z}_p^*$ we denote the multiplicative group of integers in which multiplications are computed modulo the prime p. With $\mathbb{Z}_p$ we denote the additive group of integers in which additions are computed modulo p. We handle negative values as $-k * x = k * (-x) = -(k * x)$ and $x^{-k} = (x^{-1})^k = (x^k)^{-1}$.

For p prime, we can define the prime-order field $(\mathbb{Z}_p, +, *, -, ^{-1}, 0, 1)$ combining the additive group $(\mathbb{Z}_p, +, -, 0)$ and the multiplicative group $(\mathbb{Z}_p^*, *, ^{-1}, 1)$ into a single algebraic structure with the additional property of distributivity of multiplication over addition ($(x+y)*z = (x*z)+(y*z)$ for any $\{x, y, z\} \subset \mathbb{Z}_p$).

Pairing A map $\theta : X \times Y \to Z$ is called a pairing if it provides

- *bilinearity* ($\theta(x_1 * x_2, y) = \theta(x_1, y) * \theta(x_2, y)$ and $\theta(x, y_1 * y_2) = \theta(x, y_1) * \theta(x, y_2)$),
- *non-degeneracy* (for all generators x and y, $\theta(x, y)$ generates Z) and
- *efficiency* (θ is efficiently computable).

We call a pairing *Type 1* if $G_1 = G_2$, *Type 2* if $G_1 \neq G_2$ but there exists a homomorphism from G_2 to G_1, and *Type 3* if $G_1 \neq G_2$ and there exists no homomorphism. [GPS08]. Implementation of pairings are feasible using the Tate or the Weil pairing, applying Miller's algorithm [Lyn07].

2.1 Cryptographic basics

Negligible A function $\epsilon : N \rightarrow [0, 1]$ is *negligible* if for all $c \geq 0$ there exists $k_c \geq 0$ such that $\epsilon(k) \leq \frac{1}{k^c}$ for all $k > k_c$ [Kat10]. We declare a cryptographic scheme as *secure* if the success probability of the attacker to reach its goal using its assigned capabilities is negligible.

Security parameter A *security parameter* describes the cryptographic security of a scheme; the amount of computational power required to break a scheme or property by a polynomially bounded adversary [Kat10]. We denote the security parameter as 1^k.

Hardness Assumptions The Discrete Log (DL) assumption states, that it is hard to find x for given $y = g^x$. The computational Diffie-Hellman (CDH) assumption states, that it is hard to compute g^{ab} from given $y = g^a$ and $z = g^b$. Further, the decisional Diffie-Hellman (DDH) states it is hard to differentiate $(g^a,\ g^b,\ g^{ab})$ and $(g^a,\ g^b,\ g^c)$. [Kat10] One can easily verify that solving DL implies solving CDH, and solving CDH implies solving DDH (while the reverse does not hold). It is assumed that DDH (and hence CDH and DL) holds in $G \subset \mathbb{Z}_p^*$ for $|G|$ prime [HKLD17].

Solving the discrete log Some algorithms solving the discrete log exist which are faster than simply bruteforcing. For p the prime group order, the best algorithms available (2015) are variants of the deterministic baby-step giant-step algorithm (succeeds in $O(\sqrt{p})$ time and space) or the probabilistic Pollard's rho (low space, $O(\sqrt{p})$ time) [GWZ].

Hash functions A *hash function* maps message strings of arbitrary size to some result set. A hash function can be modeled by a *random oracle*: One regards the hash function as a black box that responds to a query for the hash value of a bitstring by giving a random value. For each new query the oracle makes an independent random choice; while for each repeating query the same response is used. When we assume the existence of such a hash function, we are in the *random oracle model* [BR93].

2.2 Proofs of Knowledge

With a *proof of knowledge*, some *Prover* P proves knowledge of some secret x to a *Verifier* V. These proofs are *zero-knowledge* if V does not learn anything about the secret x in the process.

Implementation is feasible using a *one-way group homomorphism* which is a map $\phi : X \rightarrow Y$ for which inversion is hard (e.g. $\phi^{-1} : Y \rightarrow X$ is unknown or hard to compute). A potential instantiation of the one-way group homomorphism is to use a cyclic group G (in which DL is believed to be hard) and one of its generators $g \in G$ to define ϕ as $\phi(x) : g^x = y$.

A three-move Zero-Knowledge Proof of Knowledge using a one-way group homomorphism is presented in Protocol 1.

Publicly known is the one-way group homomorphism $\phi : X \rightarrow Y$ (in which inversion is believed to be hard) and $y \in Y$.

The *Prover* P creates a t using a random value k (note that k is called w in [HKLD17]) and ϕ and sends it to the *Verifier* V. V responds with a challenge c. P uses the secret x to generate a receipt s based on c and t. V checks if the received s indeed satisfies the correctness criteria, and outputs `accept` or `reject`. One interaction of this three-move protocol is fully defined by its triplet (t, c, s).

Prover		**Verifier**
$k \xleftarrow{\$} X$ $t = \phi(k)$		
	$\xrightarrow{\quad t \quad}$	
		$c \xleftarrow{\$} X$
	$\xleftarrow{\quad c \quad}$	
$s = k + x * c$		
	$\xrightarrow{\quad s \quad}$	
		$g^s = t * y^c$

Correctness holds because $\phi(s) = t * y^c$ equals $\phi(k + x * c) = \phi(k) * \phi(x)^c$.

Protocol 1: Zero-Knowledge Proof of Knowledge using a one-way group homomorphism.

Non-interactive Proof of Knowledge in the Random Oracle Model With the Fiat-Shamir transform, one can convert three-move interactive proofs such as Protocol 1 into a non-interactive proof in the random oracle model: Instead of V choosing the random c, P calculates $c = h(y, t)$ for some hash function h. The triplet (t, c, s) can now be produced independently of V. For verification, V needs to additionally check that indeed the expected c was used [FS87].

We illustrate this idea in Protocol 2. To make this implementation easy to integrate in other protocols, we modify the function signatures to fully define the public parameters.

Publicly known is the cyclic group G (in which DL is believed to be hard) and its generator $g \in G$, $y \in Y$ and the random oracle hash function $h : G \to G$. We include the public values into the function signature to ensure the functions do not relay on any global state.

The *Prover* P creates a proof π using **Proof** and sends it to the *Verifier* V. V checks with **Verify** if the received π indeed satisfies the correctness criteria.

Proof$(G,\ g,\ y,\ x) \to \pi$

$k \xleftarrow{\$} X$
$t = g^k$
$c = h(y, t)$
$s = k + x * c$
return $\pi = (t, s)$

Verify$(G,\ g,\ y,\ \pi) \to$ `accept` **or** `reject`

$(t, s) = \pi$
$c = h(y, t)$
iff $g^s = t * y^c$ then `accept` else `reject`.

Correctness holds because $g^s = t * y^c$ equals $g^{k+x*c} = g^k * g^{x*c}$.

Protocol 2: Non-interactive Zero-Knowledge Proof of Knowledge in the ROM model using the DL assumption.

2.3 Sigma Protocols and Properties

Effective relation An *effective relation* is a binary relation $R \subseteq X \times Y$, where X, Y and R are efficiently recognizable finite sets. Elements of Y are called statements. If $(x, y) \in R$, then x is called a *witness* for y [BS17].

Sigma Protocol Let $R \subseteq X \times Y$ be an effective relation. A *Sigma Protocol* for R is a pair (P, V):

- P is an interactive protocol algorithm called the *Prover*, which takes as input the witness statement pair $(x, y) \in R$.
- V is an interactive protocol algorithm called the *Verifier*, which takes as input a statement $y \in Y$, and which outputs `accept` or `reject`.
  - To start the protocol, P computes a message t, called the *commitment*, and sends t to V;
  - Upon receiving P's commitment t, V chooses a *challenge* c at random from a finite challenge space C, and sends c to P;

– Upon receiving V's challenge c, P computes a *response* s, and sends s to V;

– Upon receiving P's response s, V outputs either *accept* or *reject*, which must be computed strictly as a function of the statement y and the *conversation* (t, c, s). In particular, V does not make any random choices other than the selection of the challenge - all other computations are completely deterministic.

We require that for all $(x, y) \in R$, when $P(x, y)$ and $V(y)$ interact with each other, $V(y)$ always outputs *accept* [BS17].

Knowledge soundness Let (P, V) be a Sigma protocol for $R \subseteq X \times Y$. We say that (P, V) provides knowledge soundness if there is an efficient deterministic algorithm **Ext** (called a *witness extractor*) with the following property: Given as input a statement $y \in Y$, along with two accepting conversations (t, c, s) and (t, c', s') for y, where $c \neq c'$, algorithm **Ext** always outputs $x \in X$ such that $(x, y) \in R$ (i.e., x is a witness for y). [BS17]

Honest Verifier Zero Knowledge Let (P, V) be a Sigma protocol for $R \subseteq X \times Y$ with challenge space C. We say that (P, V) is *honest verifier zero knowledge* (HVZK), if there exists an efficient probabilistic algorithm **Sim** (called a simulator) that takes as input $y \in Y$. It satisfies that for all $(x, y) \in R$, if we compute $(t, c, s) \xleftarrow{\$} \mathbf{Sim}(y)$, then (t, c, s) has the same distribution as that of a transcript of a conversation between $P(x, y)$ and $V(y)$ [BS17].

Special Honest Verifier Zero Knowledge Let (P, V) be a Sigma protocol for $R \subseteq X \times Y$ with challenge space C. We say that (P, V) is *special honest verifier zero knowledge* (special HVZK), if there exists an efficient probabilistic algorithm **Sim** (called a simulator) that takes as input $(y, c) \in Y \times C$, and satisfies the following properties:

- for all inputs $(y, c) \in Y \times C$, algorithm **Sim** always outputs a pair (t, s) such that (t, c, s) is an accepting conversation for y;
- for all $(x, y) \in R$, if we compute $c \xleftarrow{\$} C, (t, s) \xleftarrow{\$} \mathbf{Sim}(y, c)$, then (t, c, s) has the same distribution as that of a transcript of a conversation between $P(x, y)$ and $V(y)$.

Note that special HVZK implies HVZK [BS17].

Attack game for one-way key generation Let K be a key generation for $R \subseteq X \times Y$. For a given adversary A, the attack game runs as follows:

- The challenger runs K and gets sk, pk. It then sends $pk = y$ to A;
- A outputs $\hat{x} \in X$. We say that the adversary wins the game if $(\hat{x}, y) \in R$.

We define A's advantage with respect to K, denoted $OWadv[A, K]$, as the probability that A wins the game. We say that a key generation algorithm K is *one way* if for all efficient adversaries A, the quantity $OWadv[A, K]$ is negligible.

2.4 Identification protocols

An *identification protocol* $I = (K, P, V)$ is executed between a *Prover* P and a *Verifier* V. A secret key sk and its respective public key pk are generated by K; its only argument (which we will omit for brevity from here on) is the security parameter 1^k to determine the key length. After generating the keys, P receives the secret key sk while both P and V receive the public key pk. Then a protocol is executed in which P tries to convince V it indeed holds the secret key sk.

When describing its algorithms in detail, we may also denote an non-interactive identification scheme by $I = (\textbf{KeyGen}, \textbf{Proof}, \textbf{Verify})$.

Canonical identification protocol We denote an identification protocol *canonical*, when it is executed as a characteristic three-move protocol. Denoted as $I = (K, P_t, V_c, P_s, V_v)$, after executing key generation K, it goes as follows: Commitment t is generated by P with $P_t(sk)$ and sent to V. V responds with challenge c generated by $V_c(pk, t)$. P then calculates the receipt s by $P_s(sk, t, c)$ and sends it to V. V finally executes $V_v(pk, t, c, s)$ to *accept* or *reject* the execution. Note that all Sigma protocols are canonical [AABN02].

Security notions One notion of security of identification protocols is security against impersonation: The attacker without knowing sk can not convince the verifier it is talking to the real prover. We will define security against impersonation under active attacks and security against impersonation under passive attacks. More powerful attackers have been proposed [HKL19] [BP02], but for the purpose of this work the two notions are sufficient.

Attack game for secure identification under direct attacks For a given identification protocol $I = (K, P, V)$ and a given adversary A, the attack game runs as follows:

- *Key generation phase.* The challenger runs K and sends the resulting pk to A.
- *Impersonation attempt.* The challenger and A now interact, with the challenger following the verifier's algorithm V (with input pk), and with A playing the role of a prover, but not necessarily following the prover's algorithm P (indeed, A does not receive the secret key sk).

We say that the adversary wins the game if V outputs `accept` at the end of the interaction. We define A's advantage with respect to I, denoted $ID1adv[A, I]$, as the probability that A wins the game. We say that an identification protocol I is *secure against direct attacks* if for all efficient adversaries

A, the quantity *ID1adv*$[A, I]$ is negligible [BS17]. We denote this notation also as *security against impersonation under active attacks* (*IMP-AA*) as it is mentioned in [AABN02].

Attack game for secure identification under eavesdropping attack For a given identification protocol $I = (K, P, V)$ and a given adversary A, the attack game runs as follows:

- *Key generation phase.* The challenger runs K and sends the resulting *pk* to A.
- *Eavesdropping phase.* The adversary requests some number, say q, of transcripts of conversations between P and V. The challenger complies by running the interaction between P and V a total of q times, each time with P initialized with input *sk* and V initialized with *pk*. The challenger sends these transcripts $T_1, \ldots, T_q$ to the adversary.
- *Impersonation attempt.* The challenger and A now interact, with the challenger following the verifier's algorithm V (with input *pk*), and with A playing the role of a prover, but not necessarily following the prover's algorithm P (indeed, A does not receive the secret key *sk*).

We say that the adversary wins the game if V outputs `accept` at the end of the interaction. We define A's advantage with respect to I, denoted *ID2adv*$[A, I]$, as the probability that A wins the game. We say that an identification protocol I is *secure against eavesdropping attacks* if for all efficient adversaries A, the quantity *ID2adv*$[A, I]$ is negligible [BS17]. We denote this notation also as *security against impersonation under passive attacks* (*IMP-PA*) as it is mentioned in [AABN02].

2.5 Signature schemes

A *signature scheme* $S = (K, P, V)$ is executed between a *Prover* P and a *Verifier* V. A secret key *sk* and its respective public key *pk* is generated with by K; its only argument (which we will omit for brevity from here on) is the security parameter 1^k to determine the key length. After generating the keys, P receives the secret key *sk* while both P and V receive the public key *pk*. P produces a signature π of a message m using *sk*. V then verifies the validity of π relative to the message m and the *pk*.

When describing its algorithms in detail, we may also denote a signature scheme by $S = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$; for **Sign** executed by P and **Verify** by V.

Security notions One notion of security of signature schemes is unforgeability: The attacker, without knowing *sk*, can not produce a signature V `accepts`. We introduce a single attacker which is enough for the purpose of this work.

Attack game for secure signature under chosen message attack (existential forgery) For a given signature scheme $S = (K, P, V)$, over messages M and signatures Σ, and a given adversary A, the attack game runs as follows:

- The challenger runs K and sends the resulting pk to A.
- A queries the challenger several times. For $i = 1, 2, \ldots,$ the ith signing query is a message $m_i \in M$. Given m_i, the challenger computes $\pi_i \in P(sk, m_i)$ and then gives π_i to A.
- Eventually A outputs a candidate forgery pair $(m, \pi) \in M \times \Sigma$.

We say that the adversary wins the game if $V(pk, m, \pi) = \texttt{accept}$, and m is new, namely $m \notin m_1, m_2, \ldots$. We define A's advantage with respect to P, denoted $SIGadv[A, P]$, as the probability that A wins the game. We say that a signature scheme S is *secure* if for all efficient adversaries A, the quantity $SIGadv[A, P]$ is negligible [BS17]. We denote this notation also as *security against existential forgery under chosen message attacks* (*EUF-CMA*) as it is defined in [Kat10].

3 Authentication in *CHVote*

We provide a high level overview of *CHVote* in general and a more detailed view of the authentication mechanism. This allows us to understand the context the *Protocol* was proposed for, and ensures our analysis will be useful for *CHVote*.

Adversary model *CHVote* assumes covert adversaries in *6.2. Adversary Model and Trust Assumptions*. These adversaries are able to deviate from the protocol, but only if their attempt is likely to remain undetected. Notably, the probability to remain undetected does not have to be negligible [AL07].

Hardness assumptions of *CHVote* To construct the *CHVote* Protocol, some hardness assumptions are already taken. It includes the Random Oracle Model (ROM, e.g. for the non-interactive Zero-Knowledge Proof of Knowledge), the Decisional Diffie Hellmann (DDH, e.g. ElGamal Encryption), the Chosen-Target Computational Diffie Hellmann (CT-CDH, e.g. the OT Scheme), the gap-CDH (e.g. the key encapsulation algorithm; introduced as solving CDH with access to a DDH oracle) and the Discrete Log (DL, e.g. Pederson Commitment) assumption.

3.1 Authentication within *CHVote*

Requirements of authentication Besides other roles, *CHVote* knows multiple *Voters*, multiple *Election Authorities* and the single fully trusted *Printing Authority*. The *Voter* identifies itself to each *Election Authority* using information it received from the *Printing Authority*. Naturally, the voter must not be impersonated by the covert adversary.

Overview about authentication For each *Voter*, each *Election Authority* E_i generates a pair of public and private keys (sk_i, pk_i). It sends the sk_i to the *Printing Authority*, which combines all of them into the single $sk = \prod sk_i$. This sk is then sent to the *Voter* on a secure channel, and subsequently used for authentication. Further, each E_i broadcasts their pk_i to all other $j \neq i$ E_j. Each E_j combines all received public keys into $pk = \prod pk_j$. Note that at the end of this process, each *Election Authority* has the full public key of each *Voter* (but only a share of the corresponding private key). Each *Voter* ends up with a single full private key.

This process is done twice, resulting in sk_a and sk_b for each *Voter*. The first sk_a is used to authenticate the vote, the second sk_b to authenticate the vote confirmation. Both the vote and the vote confirmation consist of a single authenticated message which is broadcast to each election authority. Each election authority only accepts the first authenticated vote and the first authenticated vote confirmation.

3.2 Authenticating messages

The two authenticated messages themselves are multiple combined together non-interactive zero-knowledge proofs of knowledge. Non-interactivity is archived using the Fiat-Shamir transformation, and combination is done with an AND-compostion: The same $c = h(\boldsymbol{y}, \boldsymbol{t})$ is used for all combined proofs (for $\boldsymbol{y} = (y_0, y_1, ...)$ and $\boldsymbol{t} = (t_0, t_1, ...)$ the inputs of the different proofs) [HKLD17].

We describe an example of this construct in Protocol 3.

Security Guarantees [HKLD17] does not explicitly specify which identification protocol property it requires (neither in *5.4. Non-Interactive Preimage Proofs* where Schnorr's Identification protocol is introduced, nor in *6.4.6. Voter Identification* where it is described for what the protocol is used). In the verifiability analysis it is simply referred to as a "standard" signature scheme [BCG+18].

From literature, we know that Schnorr's identification protocol is proven to be secure against impersonation under concurrent attacks in [BP02].

Length of the secret The length of sk is required to be at least double the required security parameter (to offset the root in runtime of the fastest currently known DL-breaking algorithms).

3.3 Proposed non-interactive identification protocol

A pairing-based non-interactive identification protocol is proposed (Protocol 4).

Length of the secret in the *Protocol* The length of x is proposed to be chosen equal to the required security parameter. Then the group size p is chosen to be of at least double the security parameter (to offset the root in runtime of the fastest currently known DL-breaking algorithms).

Assume we are in possession of a private authentication credential sk for publicly known $pk = g^{sk}$ (like Schnorr Identification [Sch90]). We want to authenticate some other private value a for publicly known $b = g^a$. Note how the scheme can be easily extended to support multiple values to authenticate at the same time.

Publicly known is the cyclic group G (in which DL is believed to be hard) of order p and one of its generators $g \in G$. Further, we assume the random oracle hash function h mapping arbitrary input to G.

Proof(pk, b, sk, a) $\rightarrow \pi$

$w_1 \xleftarrow{\$} \mathbb{Z}_p$
$w_2 \xleftarrow{\$} \mathbb{Z}_p$

$t_1 = g^{w_1}$
$t_2 = g^{w_2}$
$t = (t_1, t_2)$

$y = (pk, b)$
$c = h(y, t)$

$s_1 = w_1 - c * sk$
$s_2 = w_2 - c * a$
$s = (s_1, s_2)$

return $\pi = (t, s)$.

Verify(pk, b, π) $\rightarrow$ `accept` **or** `reject`

$(t, s) = \pi$
$(t_1, t_2) = t$
$(s_1, s_2) = s$

$y = (pk, b)$
$c = h(y, t)$
iff $t_1 \neq pk^c * g^{s_1}$ then `reject`
iff $t_2 \neq b^c * g^{s_2}$ then `reject`

return `accept`

Correctness holds because $pk^c * g^{s_1} = g^{sk*c} * g^{w_1 - c*sk} = g^{w_1} = t_1$ and $b^c * g^{s_2} = g^{a*c} * g^{w_2 - c*a} = g^{w_2} = t_2$.

Protocol 3: Example of authenticating values in *CHVote*.

Anticipated changes in *CHVote* when implementing the *Protocol* The new *Protocol* would conceptually work very similar: When and how messages are exchanged does not need change (although the public key and the signature now each consist of two values). The *Printing Authority* needs no changes at all; key aggregation works the same as before. The *Voter* and the *Voting Authorities* need some adaptations (including how the message authentication is performed). We forgo a detailed discussion for the purpose of this work.

Publicly known is the cyclic group G (in which DL is believed to be hard) of order p and its independent generators $g_1 \in G$ and $g_2 \in G$, the bilinear pairing $\theta : G \times G \to H$ (for some arbitrary H), and the maximal length k of the private key, for $k \leq |p|$. We use Protocol 2 as a subprotocol.

Keys are generated with **KeyGen**; the *Prover* P receives the secret key x and the *Verifier* V receives the public key $\hat{x}$. P uses **Proof** to generate the receipt $\hat{x}'$ and two proofs π_1 and π_2 (which show P generated $\hat{x}'$ using knowledge of x). The three values are then send to V, which verifies its validity with **Verify**.

KeyGen() $\to (x, \hat{x})$

$x \xleftarrow{\$} \mathbb{Z}_{2^k}$

$r_1 \xleftarrow{\$} \mathbb{Z}_p$
$r_2 = r_1 + x$

$\hat{x_1} = {g_1}^{r_1}$
$\hat{x_2} = {g_2}^{r_2}$

return x and $\hat{x} = (\hat{x_1}, \hat{x_2})$.

Proof(x**)** $\to (\pi_1, \pi_2, \hat{x}')$

$r'_1 \xleftarrow{\$} \mathbb{Z}_p$
$r'_2 = r'_1 + x$

$\hat{x_1}' = {g_1}^{r'_1}$
$\hat{x_2}' = {g_2}^{r'_2}$

$\pi'_1 =$ **2.Proof**$(G,\ g_1,\ \hat{x_1}',\ r'_1)$
$\pi'_2 =$ **2.Proof**$(G,\ g_2,\ \hat{x_2}',\ r'_2)$

return π'_1, π'_2 and $\hat{x}' = (\hat{x_1}', \hat{x_2}')$

Verify($\hat{x}$**,** π'_1**,** π'_2**,** $\hat{x}'$**)** $\to$ `accept` **or** `reject`

$(\hat{x_1}, \hat{x_2}) = \hat{x}$
$(\hat{x_1}', \hat{x_2}') = \hat{x}'$

iff not **2.Verify**$(G,\ g_1,\ \hat{x_1}',\ \pi'_1)$ then `reject`
iff not **2.Verify**$(G,\ g_2,\ \hat{x_2}',\ \pi'_2)$ then `reject`

$c_1 = \theta(\frac{\hat{x_1}}{\hat{x_1}'}, g_2)$
$c_2 = \theta(g_1, \frac{\hat{x_2}}{\hat{x_2}'})$
iff $c_1 == c_2$ then `accept` else `reject`

Protocol 4: Pairing-based non-interactive identification protocol.

4 A pairing-based identification protocol

We now use Protocol 4 to motivate Protocol 5: Transformed to its interactive equivalent, we can apply well-established transformations.

4.1 Sigma protocol

Theorem 1 *Protocol 5 is a sigma protocol.*

Proof. The prover algorithm P and verifier algorithm V form the (P, V) pair of the sigma protocol for the relation $R \subseteq X \times Y$, where

$$X = \mathbb{Z}_p \times \mathbb{Z}_{2^k}, Y = G \times G$$
$$R = \{((r_1, x), (\hat{x_1}, \hat{x_2})) : g_1^{r_1} = \hat{x_1} \wedge g_2^{r_1+x} = \hat{x_2} \wedge \theta(\frac{\hat{x_1}}{\hat{x_1}'}, g_2) = \theta(g_1, \frac{\hat{x_2}}{\hat{x_2}'})\} \tag{1}$$

The last part holds due to the bilinearity of θ:

$$\begin{aligned}
\theta(\frac{\hat{x_1}}{\hat{x_1}'}, g_2) &= \theta(g_1, \frac{\hat{x_2}}{\hat{x_2}'}) \\
\theta(g_1, g_2)^{r_1 - r_1'} &= \theta(g_1, g_2)^{r_2 - r_2'} \\
\theta(g_1, g_2)^{r_1 - r_1'} &= \theta(g_1, g_2)^{r_1 + x - (r_1' + x)} \\
\theta(g_1, g_2)^{r_1 - r_1'} &= \theta(g_1, g_2)^{r_1 - r_1'}
\end{aligned} \tag{2}$$

It is obvious the rest of the Sigma definition is also fulfilled. □

4.2 IMP-AA identification protocol

To prove Protocol 5 secure against impersonation under active attacks, we apply the following theorem:

Theorem 2 (19.14 from [BS17]) *Let (P, V) be a Sigma protocol for an effective relation R with a large challenge space. Let K be a key generation algorithm for R. If (P, V) provides knowledge soundness and K is one-way, then the identification scheme $I = (K, P, V)$ is secure against direct attacks. [BS17]*

In particular, suppose A is an efficient impersonation adversary attacking I via a direct attack (see section 2.4), with advantage $\epsilon = ID1adv[A, I]$. Then there exists an efficient adversary B attacking G via K (see section 2.3) (whose running time is about twice that of A), with advantage $\epsilon' = OWadv[B, K]$, such that

$$\epsilon' \geq \epsilon^2 - \epsilon/N, \tag{3}$$

where N is the size of the challenge space, which implies

$$\epsilon \leq \frac{1}{N} + \sqrt{\epsilon'} \tag{4}$$

Publicly known is the cyclic group G (in which DL is believed to be hard) of order p and its independent generators $g_1 \in G$ and $g_2 \in G$, the bilinear pairing $\theta : G \times G \to H$ (for some arbitrary H), and the maximal length k of the private key, for $k \leq |p|$.

KeyGen of Protocol 4 is used to generate the keys. The *Prover* P receives the secret key x and the *Verifier* V receives the public key $\hat{x}$. Then the protocol is executed follows:

Prover		**Verifier**
$r'_1 \xleftarrow{\$} \mathbb{Z}_p$ $r'_2 = r'_1 + x$ $\hat{x_1}' = g_1^{r'_1}$ $\hat{x_2}' = g_2^{r'_2}$		
....	....	....
$k_1 \xleftarrow{\$} \mathbb{Z}_p$ $k_2 \xleftarrow{\$} \mathbb{Z}_p$ $t_1 = g_1^{k_1}$ $t_2 = g_2^{k_2}$		
	$\xrightarrow{\hat{x_1}', \hat{x_2}', t_1, t_2}$	
		$c \xleftarrow{\$} \mathbb{Z}_p$
	$\xleftarrow{c}$	
$s_1 = k_1 + r'_1 * c$ $s_2 = k_2 + r'_2 * c$		
	$\xrightarrow{s_1, s_2}$	
		$g_1^{s_1} = t_1 * \hat{x_1}'^{c}$ $g_2^{s_2} = t_2 * \hat{x_2}'^{c}$
....	....	....
		$\theta(\frac{\hat{x_1}}{\hat{x_1}'}, g_2) = \theta(g_1, \frac{\hat{x_2}}{\hat{x_2}'})$

Protocol 5: Pairing-based identification protocol.

Knowledge soundness Given two accepting conversations (t, c, s) and (t, c', s') (with $c \neq c'$) for $(\hat{x_1}, \hat{x_2}) \in Y$, **Ext** for Protocol 5 can output $(r_1, x) \in X$ such that $((r_1, x), (\hat{x_1}, \hat{x_2})) \in R$ as follows:

$$\begin{aligned} r_1' &= \frac{s_1 - s_1'}{c - c'} = \frac{k_1 + r_1' * c - k_1 - r_1' * c'}{c - c'} = \frac{(c - c') * r_1'}{c - c'} \\ r_2' &= \frac{s_2 - s_2'}{c - c'} = \frac{k_2 + r_2' * c - k_2 - r_2' * c'}{c - c'} = \frac{(c - c') * r_2'}{c - c'} \\ \text{output } &(r_1 = r_1', x = r_2' - r_1') \end{aligned} \tag{5}$$

One-way KeyGen Note that **KeyGen** of Protocol 5 computes $sk = x \xleftarrow{\$} \mathbb{Z}_{2^k}$ and $pk = (\hat{x_1} = {g_1}^{r_1}, \hat{x_2} = {g_2}^{r_2})$ for $r_1 \xleftarrow{\$} \mathbb{Z}_p$ and $r_2 = r_1 + x$. It is obvious that the computation of pk out of sk is one-way under the DL assumption.

Theorem 3 *Under the DL assumption for $\mathbb{G}$, and assuming $N = |\mathbb{Z}_p|$ is super-poly, Protocol 5 is secure against active attacks.*

Proof. We have shown in Theorem 1 that Protocol 5 is a Sigma protocol. Together with our proofs of knowledge soundness and of the one-way **KeyGen**, we have shown all preconditions of Theorem 2, and our claim follows. □

4.3 IMP-PA identification protocol

To prove our identification protocol secure against impersonation under passive attacks, we apply the following theorem:

Theorem 4 (19.3 from [BS17]) *If an identification protocol I is secure against direct attacks, and is HVZK, then it is secure against eavesdropping attacks. [BS17]*

> *In particular, if I is HVZK with simulator **Sim**, then for every impersonation adversary A that attacks I via an eavesdropping attack (see section 2.4) obtaining up to q transcripts, there is an adversary B that attacks I via a direct attack (see section 2.4) where B is an elementary wrapper around A (and where B runs **Sim** at most Q times), such that*
>
> $$\mathit{ID2adv}[A, I] = \mathit{ID1adv}[B, I] \tag{6}$$

Special HVZK To show that Protocol 5 is special HVZK, we show that we are able to generate transcripts t' of a successful protocol run without knowing the secret x given the challenge $c \xleftarrow{\$} C$. These generated transcripts are perfect indistinguishable from real transcripts as we can show that their probability distributions are exactly the same.

In our original protocol, the distribution of the values in a protocol run ($\hat{x_1}'$, $\hat{x_2}'$, t_1, t_2, c, s_1, s_2) are as follows:

- $\hat{x_1}'$ is calculated by raising the uniform random r_1 to the generator g_1.
- $\hat{x_2}'$ is calculated by raising the uniform random r_1 plus a secret x to the generator g_2.
- t_1 is calculated by raising the uniform random k_1 to the generator g_1. Same for t_2, for the uniform random k_2 and the generator g_2.
- s_1 is calculated by adding the uniform random k_1 to another value. [2]. Same for s_2, by adding the uniform random k_2.

We summarize that all values except $\hat{x_2}'$ are distributed uniformly at random: Either adding a uniform random or raising a uniform random to a generator leads to a uniform random distribution.

We avoid describing the distribution of $\hat{x_2}'$ exactly, but rather observe the following: The distribution of $\hat{x_2}'$ relates to the distribution of $\hat{x_1}'$ the same way as the distribution of the two values of the respective public key ($\hat{x_2}$, $\hat{x_1}$) relate to each other. Note that this is due to the fact as **KeyGen** generates $\hat{x_1}$ and $\hat{x_2}$ in the same way as the *Prover* generates $\hat{x_1'}$ and $\hat{x_2}$.

To generate a transcript without knowing x, replicating the observed distributions of a real transcript given the challenge $c \overset{\$}{\leftarrow} C$, we propose to choose the values as follows:

$$r', s_1, s_2 \overset{\$}{\leftarrow} \mathbb{Z}_p$$

$$\hat{x_1}' = \hat{x_1} * g_1^{r'}$$
$$\hat{x_2}' = \hat{x_2} * g_2^{r'}$$

$$t_1 = g_1^{s_1} * \hat{x_1}'^{-c}$$
$$t_2 = g_2^{s_2} * \hat{x_2}'^{-c}$$

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

- $\hat{x_1}'$ is distributed uniformly at random: As r' is chosen uniformly at random, using it as an exponent for g_1 results in a uniform random value. Multiplying it to $\hat{x_1}$ "shifts" $\hat{x_1}$ by the uniform random amount, resulting in a uniform at random distribution.
- $\hat{x_2}'$ replicates the specific distribution relative to $\hat{x_1'}$ as observed in a real protocol run: We use the same uniform random r' as an exponent for g_2 and multiply this to $\hat{x_2}$, hence preserving the relative distribution between $\hat{x_2}$ and $\hat{x_1}$ for $\hat{x_2}'$ and $\hat{x_1}'$.

[2]Note that k_1 was already used to calculate t_1. This however does not produce any useful insight for the attacker, as these calculations are performed in the group $\mathbb{G}$ in which the hardness of the discrete log is assumed.

- 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 $\hat{x_1}'^{-c}$ is also uniform random following the same argumentation. Multiplying two uniform random values results in a uniform at random distribution. The same argumentation applies to t_2, with s_2, g_2 and $\hat{x_1}'$.

- c, s_1 and s_2 are each distributed uniformly at random as each is chosen uniformly at random.

As a final step, we need to ensure the generated transcript also passes our correctness checks. There are three checks performed; two verifying the relationship between t_1, c and s_1 (respectively t_2, c, s_2) and one involving our pairing:

$$\begin{aligned} g_1^{s_1} = t_1 * \hat{x_1}'^{c} \qquad &= \qquad g_1^{s_1} = g_1^{s_1} * \hat{x_1}'^{-c} * \hat{x_1}'^{c} \\ g_2^{s_2} = t_2 * \hat{x_2}'^{c} \qquad &= \qquad g_2^{s_2} = g_2^{s_2} * \hat{x_2}'^{-c} * \hat{x_2}'^{c} \end{aligned} \tag{7}$$

$$\begin{aligned} \theta(\frac{\hat{x_1}}{\hat{x_1}'}, g_2) &= \theta(g_1, \frac{\hat{x_2}}{\hat{x_2}'}) \\ \theta(\frac{\hat{x_1}}{\hat{x_1} * g_1^{r'}}, g_2) &= \theta(g_1, \frac{\hat{x_2}}{\hat{x_2} * g_2^{r'}}) \\ \theta(g_1, g_2)^{-r'} &= \theta(g_1, g_2)^{-r'} \end{aligned} \tag{8}$$

HVZK Protocol 5 is also HVZK, as this directly follows from special HVZK.

Theorem 5 *Under the DL assumption for $\mathbb{G}$, and assuming $N = |\mathbb{Z}_p|$ is super-poly, Protocol 5 is secure against passive attacks.*

Proof. From Theorem 3 we know Protocol 5 is secure against active attacks. Together with our HVZK proof, we fulfill all preconditions to apply Theorem 4 and our claim follows. □

4.4 EUF-CMA signature scheme

Using a a construction proposed by [AABN02] (the Generalized Fiat-Shamir Transform, see Construction 1) we can transform an IMP-PA secure identification protocol into an EUF-CMA secure signature scheme.

Theorem 6 (3.3 from [AABN02]) *Let ID be a non-trivial, canonical identification protocol, and let S be the associated signature scheme as per Construction 1 with $s(1^k) = 0$. Then S is polynomially-secure against chosen-message attacks in the random oracle model if and only if ID is polynomially-secure*

Let $ID = (K, P_t, V_c, P_s, V_v)$ be a canonical identification protocol and let $s : N \to N$ be a function which we call the seed length. We associate to these a digital signature scheme $S = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$. For **KeyGen**, it uses K of the identification protocol. Let h be a hash function mapping arbitrary input to G and which output length equals the challenge length of the identification protocol.

Note that the signing algorithm is randomized, using a random tape whose length is $s(1^k)$ plus the length of the random tape of the prover. Furthermore, the chosen random seed is included as part of the signature to make verification possible.

Sign(sk, m) $\to \pi$

$R \xleftarrow{\$} 0, 1^{s(k)}$

$t = P_t(sk)$

$c = h(R, t, m)$

$s = P_s(sk, t, c)$

return $\pi = (R, t, s)$.

Verify(pk, m, π) $\to$ `accept` **or** `reject`

$(R, t, s) = \pi$

$c = h(R, t, m)$

return $V_v(pk, t, c, s)$.

Construction 1: Generalized Fiat-Shamir Transform

against impersonation under passive attacks.

We clarify all terminology used which we have not introduced yet, or introduced differently:

- *non-trivial* requires that for commitments drawn uniform at random from a set, said set must have super-polynomial size (by Definition 3.2 of [AABN02]).
- *secure against chosen-message attacks* refers to EUF-CMA, implied by the security game introduced in Definition 2.2 in [AABN02]: The forger is able to pick the message it intends to forge, and a forgery is accepted if the message has not been queried at the signing oracle yet.

Before applying the transformation, we first simplify it: We can remove R as it is always 0 by Theorem 6. We arrive at Construction 2.

Now we apply Construction 2 to Protocol 5 which results in Protocol 6.

Theorem 7 *Under the DL assumption for $\mathbb{G}$, and assuming $N = |\mathbb{Z}_p|$ is super-poly, Protocol 6 is EUF-CMA secure.*

Proof. We have shown for Protocol 5 that it is secure against impersonation under active attacks in Theorem 5. We easily verify the scheme is both non-trivial (as the challenge space is super-poly) and a canonical scheme (follows

Let $ID = (K, P_t, V_c, P_s, V_v)$ be a canonical identification protocol and let $s : N \to N$ be a function which we call the seed length. We associate to these a digital signature scheme $S = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$. For **KeyGen**, it uses K of the identification protocol. Let h be a hash function mapping arbitrary input to G and which output length equals the challenge length of the identification protocol.

Sign$(sk,\ m) \to \pi$

$t = P_t(sk)$
$c = h(t, m)$
$s = P_s(sk, t, c)$
return $\pi = (t, s)$.

Verify$(pk,\ m,\ \pi) \to$ `accept` **or** `reject`

$(t, s) = \pi$
$c = h(t, m)$
return $V_v(pk, t, c, s)$.

Construction 2: Simplified Fiat-Shamir Transform

from Theorem 1). Hence we fulfilled all preconditions of Theorem 6. Applying the Construction 2 results in Protocol 6 and hence our claim follows. □

4.5 Pairing implementation notes

For the purpose the protocol is defined for, the most important requirement is the short secret key length while computational effort is not of a big concern. Still, improving performance is beneficial: A faster *Prover* might improve usability (as the voting client of the voter reacts faster), while an improved **KeyGen** and *Verifier* might allow the system to support more voters.

Our Protocol 5 and all its derivations use the pairing θ only for the following two operations in the *Verifier*:

$$c_1 = \theta(\frac{\hat{x_1}}{\hat{x_1}'}, g_2), c_2 = \theta(g_1, \frac{\hat{x_2}}{\hat{x_2}'}) \tag{9}$$

Protocol 5 never mixes numbers that involve g_1 with numbers that involve g_2: Assuming $g_1 \in G_1$ and $g_2 \in G_2$, we do not require a homomorphism between G_1 and G_2. We observe that we do not necessarily need to use a *Type 1* pairing, but may also choose a *Type 2* or *Type 3* pairing.

Besides θ, we need a single operation per generator for each **KeyGen**, *Prover* and *Verifier*. There exists a *Type 3* pairing with efficient group operations in G_1, k^2-less-efficient group operations in G_2 and an efficient θ for a wide flexibility of system parameters. We might also opt for *Type 1* pairing (with efficient group operations in G_2), but its performance degrades fast with higher security levels, and it is less flexible concerning system parameters [GPS08]

If we use elliptic curves to implement our pairing, we require its group size to be 2^{2k} for the security parameter k. [GPS08] Hence for the 128 bits security that *CHVote* requires at its highest security level 3, we need to choose a key size of 256 bits.

Publicly known is the cyclic group G (in which DL is believed to be hard) of order p and its independent generators $g_1 \in G$ and $g_2 \in G$, the bilinear pairing $\theta : G \times G \to H$ (for some arbitrary H), the maximal length of the private key k, for $k \leq p$, and the hash function h with its output length equal to the length of r_1 resp. r_2.

Keys are generated with **KeyGen**; the *Prover* P receives the secret key x and the *Verifier* V receives the public key $\hat{x}$. P uses **Sign** to generate the signature $\hat{x}'$ and two receipts π_1 and π_2 (which show P generated $\hat{x}'$ using knowledge of x). The three values are then send to V, which verifies its validity with **Verify**.

KeyGen() $\to (x, \hat{x})$

$x \xleftarrow{\$} \{0, \ldots, 2^k - 1\}$

$r_1 \xleftarrow{\$} \mathbb{Z}_p$
$r_2 = r_1 + x$

$\hat{x_1} = {g_1}^{r_1}$
$\hat{x_2} = {g_2}^{r_2}$

return x and $\hat{x} = (\hat{x_1}, \hat{x_2})$.

Sign$(x,\, m) \to (\pi_1, \pi_2, \hat{x}')$

$r_1' \xleftarrow{\$} \mathbb{Z}_p$
$r_2' = r_1' + x$

$k_1 \xleftarrow{\$} \mathbb{Z}_p$
$k_2 \xleftarrow{\$} \mathbb{Z}_p$
$t_1 = g_1^{k_1}$
$t_2 = g_2^{k_2}$
$c = h(m, t_1, t_2)$
$s_1 = k_1 + c * r_1'$
$s_2 = k_2 + c * r_2'$
$\pi_1 = (t_1, s_1)$
$\pi_2 = (t_2, s_2)$

$\hat{x_1}' = {g_1}^{r_1'}$
$\hat{x_2}' = {g_2}^{r_2'}$
$\hat{x}' = (\hat{x_1}', \hat{x_2}')$

return π_1, π_2 and $\hat{x}'$

Verify$(\hat{x},\, m,\, \pi_1,\, \pi_2,\, \hat{x}') \to$ `accept` **or** `reject`

$(\hat{x_1}, \hat{x_2}) = \hat{x}$
$(\hat{x_1}', \hat{x_2}') = \hat{x}'$

$(t_1, s_1) = \pi_1$
$(t_2, s_2) = \pi_2$
$c = h(m, t_1, t_2)$
iff $g_1^{s_1} \neq t_1 * \hat{x_1}'^{c}$ then `reject`
iff $g_2^{s_2} \neq t_2 * \hat{x_2}'^{c}$ then `reject`

$c_1 = \theta(\frac{\hat{x_1}}{\hat{x_1}'}, g_2)$
$c_2 = \theta(g_1, \frac{\hat{x_2}}{\hat{x_2}'})$
iff $c_1 \neq c_2$ then `reject`

return `accept`

Protocol 6: Pairing-Based Signature Scheme.

5 Alternatives

We want the authentication protocol to have a small secret size (optimally equal the security parameter) and need it to support key aggregation. We present other promising ideas; archiving the same security while using fewer resources.

Pseudo-Random Generator We could use a pseudo-random generator (PRG) (in the sense defined in [Gol09]) to map a short secret (with length equal to the security parameter) to a sufficiently long number (matching the group size requirement due to the DL-breaking algorithms).

As we need to support key aggregation, we require a key-homomophic PRG (e.g. the PRG P must support $P(x_1 + x_2) = P(x_1) + P(x_2)$).

[Negative Result] Modified Schnorr Signature Modifying the Schnorr Signature to include the core idea of Protocol 5 - to use as the exponent a large random value plus a short secret - results in Protocol 7. The pairing is gone and key aggregation works, but security relative to the DL-breaking algorithms is not improved: Given the public key $(r, y = g^{r+x})$, we can easily extract g^x.

Publicly known is the cyclic group G (in which DL is believed to be hard) and one of its generators $g \in G$.

Keys are generated with **KeyGen**; the *Prover* P receives the secret key sk and both P and the *Verifier* V receive the public key pk. P signs a message m with **Sign** to get the signature σ. V can verify σ with **Verify**.

KeyGen() $\rightarrow (sk, pk)$

$r \xleftarrow{\$} \mathbb{Z}_p$
$x \xleftarrow{\$} \{0, \dots, 2^k - 1\}$
$y = g^{r+x}$
return $sk = x$ and $pk = (r, y)$.

Sign(sk, pk, m) $\rightarrow \sigma$

$x = sk$
$(r, y) = pk$
$k \xleftarrow{\$} \mathbb{Z}_p$
$t = g^k$
$c = h(m, t)$
$s = k - (r + x) * c$
return signature $\sigma = (c, s)$.

Verify(pk, σ, m) $\rightarrow$ `accept` or `reject`

$(c, s) = \sigma$
$c' = h(m, pk^c * g^s)$
iff $c == c'$ then `accept` else `reject`

Correctness holds because $pk^c * g^s = g^{(r+x)*c} * g^{k-(r+x)*c} = g^k$.

Protocol 7: Modified Schnorr Signature Scheme.

[Negative result] Replacing the pairing Note that the pairing in Protocol 5 is only used to proof that two exponents being raised to different generators are equal (the exponent being $r'_1 - r_1$). This property can also be shown without pairings as demonstrated in Protocol 8 [Dam10]. Arguably, not using a pairing is an improvement on the protocol as it likely reduces the complexity, implementation burden and runtime cost of *CHVote* (Protocol 5 is the first time *CHVote* considers using pairings).

Publicly known is the cyclic group G (in which DL is believed to be hard) of order p and its independent generators $g_1 \in G$ and $g_2 \in G$. Further, publicly known are $h_1 \in G$ and $h_2 \in G$. The *Prover* P knows x such that $g_1 = h_1^x$ and $g_2 = h_2^x$.

Prover		**Verifier**
$k \xleftarrow{\$} \mathbb{Z}_p$ $t_1 = g_1^k$ $t_2 = g_2^k$		
	$\xrightarrow{t_1, t_2}$	
		$c \xleftarrow{\$} \mathbb{Z}_p$
	$\xleftarrow{c}$	
$s = k + x * c$		
	$\xrightarrow{s}$	
		$g_1^s = t_1 * {h_1}^c$ $g_2^s = t_2 * {h_2}^c$

Correctness holds because $g_1^s = t_1 * {h_1}^c$ equals $g_1^{k+x*c} = g_1^k * {g_1}^{x*c}$ and $g_2^s = t_2 * {h_2}^c$ equals $g_2^{k+x*c} = g_2^k * {g_2}^{x*c}$

Protocol 8: Proof equality of exponent raised to two different generators.

However, integration of Protocol 8 into Protocol 5 fails: Neither the public key nor the private key include r_1 (the public key only includes $g_1^{r_1}$). Extending the public key with r_1 decreases the security relative to the DL-breaking algorithms as already seen with Protocol 7. Further, we do not want to append r_1 to the private key, as then the private key is no longer short.

Publicly known is the cyclic group G (in which DL is believed to be hard) of order p and its independent generators $g_1 \in G$ and $g_2 \in G$, and the maximal length k of the private key, for $k \leq |p|$.

KeyGen of Protocol 4 is used to generate the keys. The *Prover* P receives the secret key x and the *Verifier* V receives the public key $\hat{x}$. Then the protocol is executed follows:

Prover		**Verifier**
$r_1' \xleftarrow{\$} \mathbb{Z}_p$ $r_2' = r_1' + x$		
$x' = r_1 - r_1'$ $h_1' = g_1^{x'}, h_2' = g_1^{x'}$		
$k' \xleftarrow{\$} \mathbb{Z}_p$ $t_1' = g_1^{k'}, t_2' = g_2^{k'}$		
....		....
$k_1 \xleftarrow{\$} \mathbb{Z}_p, k_2 \xleftarrow{\$} \mathbb{Z}_p$ $t_1 = g_1^{k_1}, t_2 = g_2^{k_2}$		
	$\xrightarrow{h_1', h_2', t_1', t_2', t_1, t_2}$	
		$c \xleftarrow{\$} \mathbb{Z}_p$
	$\xleftarrow{c}$	
$s' = k' + x' * c$ $s_1 = k_1 + r_1' * c$ $s_2 = k_2 + r_2' * c$		
	$\xrightarrow{s', s_1, s_2}$	
		$g_1^{s_1} = t_1 * \hat{x_1}'^c$ $g_2^{s_2} = t_2 * \hat{x_2}'^c$
....		....
		$g_1^{s'} = t_1' * {h_1'}^c$ $g_2^{s'} = t_2' * {h_2'}^c$

Protocol 9: Identification protocol.

References

[AABN02] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 418–433, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. doi:10.1007/3-540-46035-7_28.

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 137–156, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-70936-7_8.

[BCG+18] David Bernhard, Véronique Cortier, Pierrick Gaudry, Mathieu Turuani, and Bogdan Warinschi. Verifiability analysis of chvote. 2018.

[BP02] Mihir Bellare and Adriana Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 162–177, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany. doi:10.1007/3-540-45708-9_11.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. doi:10.1145/168588.168596.

[BS17] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Retrieved from https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_5.pdf*, 2017.

[Dam10] Ivan Damgard. On σ-protocols. *Retrieved from https://www.cs.au.dk/~ivan/Sigma.pdf*, 2010.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. doi:10.1007/3-540-47721-7_12.

[Gol09] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[GPS08] Steven D Galbraith, Kenneth G Paterson, and Nigel P Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[GWZ] Steven D Galbraith, Ping Wang, and Fangguo Zhang. Computing elliptic curve discrete logarithms with improved baby-step giant-step algorithm. *Advances in Mathematics of Communications*, 11(3):453.

[HKL19] Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 345–375, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-17659-4_12.

[HKLD17] Rolf Haenni, Reto E Koenig, Philipp Locher, and Eric Dubuis. Chvote system specification. *IACR Cryptol. ePrint Arch.*, 2017:325, 2017.

[Kat10] Jonathan Katz. *Digital signatures*. Springer Science & Business Media, 2010.

[Lyn07] Ben Lynn. *On the implementation of pairing-based cryptosystems*. PhD thesis, Stanford University Stanford, California, 2007.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards (abstract) (rump session). In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 688–689, Houthalen, Belgium, April 10–13, 1990. Springer, Heidelberg, Germany. doi:10.1007/3-540-46885-4_68.