McFly: Verifiable Encryption to the Future Made Practical

Nico Döttling[1*] Lucjan Hanzlik[1] Bernardo Magri[2] Stella Wohnig[1,3]

[1]CISPA Helmholtz Center for Information Security

[2]The University of Manchester

[3]Saarland University

{doettling,hanzlik,stella.wohnig}@cispa.de,
bernardo.magri@manchester.ac.uk

July 26, 2023

Abstract

Blockchain protocols have revolutionized the way individuals and devices can interact and transact over the internet. More recently, a trend has emerged to harness blockchain technology as a catalyst to enable advanced security features in distributed applications, in particular fairness. However, the tools employed to achieve these security features are either resource wasteful (e.g., time-lock primitives) or only efficient in theory (e.g., witness encryption). We present McFly, a protocol that allows one to efficiently "encrypt a message to the future" such that the receiver can efficiently decrypt the message at the right time. At the heart of the McFly protocol lies a novel primitive that we call signature-based witness encryption (SWE). In a nutshell, SWE allows to encrypt a plaintext with respect to a tag and a set of signature verification keys. Once a threshold multi-signature of this tag under a sufficient number of these verification keys is released, this signature can be used to efficiently decrypt an SWE ciphertext for this tag. We design and implement a practically efficient SWE scheme in the asymmetric bilinear setting. The McFly protocol, which is obtained by combining our SWE scheme with a BFT blockchain (or a blockchain finality layer) enjoys a number of advantages over alternative approaches: There is a very small computational overhead for all involved parties, the users of McFly do not need to actively maintain the blockchain, are neither required to communicate with the committees, nor are they required to post on the blockchain. To demonstrate the practicality of the McFly protocol, we implemented our SWE scheme and evaluated it on a standard laptop with Intel i7 @2,3 GHz. For the popular BLS12-381 curve, a 381-bit message and a committee of size 500 the encryption time is $9.8s$ and decryption is $14.8s$. The scheme remains practical for a committee of size 2000 with an encryption time of $58s$ and decryption time of $218s$.

*Funded by the European Union (ERC, LACONIC, 101041207). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Contents

1 Introduction **1**
- 1.1 Our Contributions . . . 2
- 1.2 Technical Overview . . . 3
- 1.3 Related Work . . . 6
- 1.4 Organisation . . . 7

2 Preliminaries **8**

3 Signature-Based Witness Encryption **13**

4 The McFly Protocol **15**
- 4.1 Formal Model and Guarantees . . . 15
- 4.2 Protocol Description . . . 18
- 4.3 Proofs . . . 18
- 4.4 Integration with Casper . . . 20
- 4.5 Applications . . . 21

5 BLS signatures with modified aggregation **22**
- 5.1 Construction . . . 22
- 5.2 Proofs . . . 23

6 Construction of Signature-based Witness Encryption **25**
- 6.1 Construction . . . 25
  - 6.1.1 Base Construction . . . 26
- 6.2 Efficiency . . . 27
- 6.3 Proofs . . . 27

7 A Compatibility Layer for Proof Systems **36**
- 7.1 Well-formedness Proofs . . . 36
- 7.2 Proofs of Plaintext Equality . . . 39
- 7.3 Putting Everything Together: Verifiable SWE . . . 41

8 Implementation and Evaluation **42**

9 Conclusion **45**

A Discussion on Proofs of Possession **48**

1 Introduction

Blockchain protocols have become increasingly popular as they revolutionized the way peer-to-peer transactions can be made. In their most basic form, blockchain protocols are run by independent parties, the so-called miners, that keep their own copy of the blockchain and verify the contents of all transactions they receive before appending them to their own copy of the blockchain. The fact that the content of the transactions can be verified *before* its inclusion in the blockchain is fundamental to the validity of the transactions and the consistency of the blockchain. However, there are many scenarios where one would like to keep the contents of a transaction secret for some time even *after* inclusion in the blockchain. One simple example is running sealed-bid auctions on the blockchain; one would like for its bid to be included in the blockchain, but at the same time such a bid should remain hidden until the end of the auction.[1] Another example that recently became very relevant with the popularization of decentralized exchanges (DEX) is the hurtful practice of transaction *frontrunning*, where malicious actors try to profit by taking advantage of possible market fluctuations that could happen after some target transaction is added to the ledger. To exploit this, the adversary tries to get its own transaction included in the ledger *before* the target transaction, by either mining the block itself and changing the order of transactions, or by offering considerably more fees for its own transaction. Hiding parts of the content of the transactions until they are final in the ledger would make it harder for adversaries to target those transactions for frontrunning. A more general application for such a mechanism, that can keep the contents of a blockchain transaction secret for some pre-defined time, would be to simply use it as a tool to realize timed-release encryption [37] without a trusted third party.

In previous works [21, 7], solutions to the problems above were based on time-lock primitives, such as time-lock puzzles (TLP) or verifiable delay functions (VDF). An inherent problem of time-lock type primitives is that they are wasteful in terms of computational resources and notoriously difficult to instantiate with concrete parameters. Usually, a reference hardware is used to measure the "fastest possible" time that it takes to solve a single operation of the puzzle (e.g., modular squaring) and this reference number is used to set the security parameters. Moreover, in a heterogeneous and decentralized system such as a blockchain, where different hardware can have gaps in speed of many orders of magnitude, an approach like this could render the system impractical. An operation that takes one time unit in the reference hardware could take 1000 time units on different hardware used in the system.

Moreover, the environmental problems that proof-of-work blockchains, where miners invest computation power to create new blocks, can cause have been intensively debated by the community and regulators. This made the majority of blockchain systems adopt a proof-of-stake (PoS) consensus for being a much more sustainable solution. In PoS systems, typically a subset of users is chosen as a committee, which jointly decides which blocks to include in the chain. This selection can be by a lottery with winning probability proportional to the amount of coins parties hold on the chain or by the parties applying by locking a relatively big amount of their coins, preventing them from spending them. In light of that, any solution employing a time-lock type primitive completely defeats the purpose of achieving a more resource-efficient and environmentally conscious system.

[1] Clearly, the auction should run on an incentive-compatible transaction ledger, where transactions paying the required fees are guaranteed to be included in the ledger within some fixed time.

1.1 Our Contributions

In that vein, we diverge from the time-lock primitive approach and propose McFly, an efficient protocol to keep the contents of a message (e.g., a blockchain transaction) secret for some pre-specified time period. McFly is based on a new primitive that we call signature witness encryption (SWE), that combined with a byzantine fault tolerance (BFT) blockchain or with any blockchain coupled with a finality layer such as Ethereum's Casper [15] or Afgjort [22] allows users to encrypt messages to a future point in time by piggybacking part of the decryption procedure on the tasks already performed by the underlying committee of the blockchain (or the finality layer) - namely voting for and signing blocks. In BFT blockchains this happens for every new potential block to reach consensus, while in a finality layer this is done for blocks at regular intervals to make them "final". We detail our contributions next.

Signature Witness Encryption We formally define a new primitive that we call signature-based witness encryption (SWE). To encrypt a message m, the encryption algorithm takes a set of verification keys for a (potentially aggregatable) multi-signature scheme[2] and a reference message r as an input and produces a ciphertext ct. The witness to decrypt ct consists of a multi-signature of the reference message r under a threshold number of keys. Note that if the receiver of such a ciphertext ct is external to the key holders, they may only observe and wait for the signatures to be made; this is sensible in a setting where we expect parties to sign the reference message naturally as a committee naturally signs blocks on a blockchain. We instantiate SWE with an aggregatable multi-signature scheme that is a BLS scheme [10] with a modified aggregation mechanism. We show, that this signature scheme fulfills the same security notions as previous aggregatable BLS multi-signatures.

Concretely, the guarantees for SWE are that (1) it correctly allows to decrypt a ciphertext given a multi-signature on the underlying reference and (2) if the adversary does not gain access to a sufficient number of signatures on the reference then ciphertext-indistinguishability holds. The security guarantee is conceptually closer related to that of identity-based encryption, rather than that of fully-fledged witness encryption; decryption is possible when a threshold number of key holders participate to unlock. We achieve this in the bilinear group setting from the bilinear Diffie-Hellman assumption. Also, unlike general witness encryption constructions [28] that are highly inefficient, we demonstrate SWE to be practicable. Furthermore, to ensure that decryption is always possible we make SWE verifiable by designing specially tailored proof systems to show well-formedness of ciphertexts as well as additional properties of the encrypted message.

McFly protocol We build an "encryption to the future" protocol by combining SWE with a BFT blockchain or a blockchain finality layer. The main idea of this is to leverage the existing committee infrastructure of the underlying blockchain that periodically signs blocks in the chain to piggyback part of the decryption procedure of the SWE scheme. At a high level, a message is encrypted with respect to a specified block height of the underlying blockchain (representing how far into the future the message should remain encrypted) and the set of verification keys of all the committee members that are supposed to sign the block at that height; once the block with the specified height is created by the committee, it automatically becomes the witness required to decrypt the ciphertext. We have the following requirements on the underlying blockchain:

- **BFT-style or finality layer.** Every (final) block in the chain must be signed by a committee of parties. These committees are allowed to be static or dynamic, with the

[2] This type of signature schemes allows to compress multiple signatures by different signers on the same messages into just one verifiable signature. In aggregatable schemes, this works even on different messages.

only requirement that the committee responsible for signing a block at a particular height must be known *some time* in advance. How much "time in advance" the committee is known is what we call horizon (following the nomenclature of [29]). For simplicity, we will explicitly assume that all blocks are immediately finalized, but our results can be easily adapted to the more general setting where the height of the next final block is known.

- **Block Structure.** We assume that blocks have a predictable header, which we will model by a block counter, and some data content. When finalizing a block the committee signs the block as usual, but additionally, it also signs the block counter separately.[3]
- **Public Key Infrastructure.** The public keys of the committee members must have a proof of knowledge. This can be achieved, e.g., by registering the keys with a PKI.
- **Honest Majority Committee.**[4] The majority of the committee behaves honestly. That is, there will not be a majority of committee members colluding to prematurely sign blocks.
- **Constant Block Production Rate.** To have a meaningful notion of "wall-clock time", the blocks must be produced at a near constant rate.

To model the requirements above, we present a blockchain functionality in Section 4.1 and later we show the security of the McFly protocol in this hybrid model. For concreteness, we informally discuss in Section 4.4 how a modified version of the currently deployed Ethereum 2.0 running with Casper "The friendly finality gadget" [15] satisfies the requirements of our blockchain functionality. Intuitively, to make Ethereum 2.0 + Casper compatible with our blockchain model we only need to add the public-key-infrastructure and require the committee members to sign a block counter separately for each finalized block. This enables encryption up to the horizon where a future committee is already known. Unfortunately, in Ethereum 2.0 this leads to a maximum horizon of 12.8 minutes. If we use "sync committees" instead, that was only introduced in Ethereum Altair [14], we can have an horizon of up to 54 hours. However, it is not clear whether sync committees enjoy the same level of trust as the standard ones.

Implementation To demonstrate the practicality of McFly, we implement the SWE scheme and run a series of benchmarks on a standard Macbook Pro with an Intel i7 processor @2,3 GHz. In Section 8 we show that for the popular BLS12-381 curve, it is possible to encrypt 381-bit messages in under 1 minute for even up to 2000 verification keys, i.e. committee size. For the same setting, decrypting takes only around 4 minutes. In the case of a supermajority threshold, the encryption time remains the same but the decryption time increases, as to be expected, to around 6 minutes. Lowering the size of the verification key set to 500 increases the efficiency. The same message can now be encrypted in 10 seconds. Depending on the threshold decryption takes 14.6 seconds for the majority of signers and 26.6 seconds for the supermajority of signers. For small committees ≤ 200 we even get encryption and decryption times smaller than 5 seconds. We stress that our results should be treated as a baseline since we used JavaScript, and any native implementation of the SWE scheme will significantly outperform our prototype.

1.2 Technical Overview

As detailed above, the key ingredient and main technical challenge of the McFly protocol is *Signature Witness Encryption* (SWE). In the following, we will provide an outline of our construction of practically efficient SWE.

[3] They use the same keys for this. This is safe whenever the underlying signature is a hash-and-sign scheme as is commonly the case.

[4] The honest majority requirement must be strengthened to honest supermajority (i.e. at least 2/3 of members being honest) if the underlying blockchain or finality layer considers a partially synchronous network model. For simplicity, we choose to describe it in the synchronous network model where honest majority plus PKI is sufficient.

SWE based on BLS Our construction of Signature-based Witness Encryption is based on the BLS signature scheme [11] and its relation to identity-based encryption [9]. Recall that BLS signatures are defined over a bilinear group, i.e. we have 3 groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ (with generators g_1, g_2, g_T) of prime-order p and an efficiently computable bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. A verification key vk is of the form $\mathsf{vk} = g_2^x$, where $x \in \mathbb{Z}_p$ is the corresponding signing key. To sign a message $T \in \{0,1\}^*$, we compute $\sigma = H(T)^x$, where $H : \{0,1\}^* \to \mathbb{G}_1$ is a hash function (which is modeled as a random oracle in the security proofs). To verify a signature σ for a message T, all we need to do is check whether $e(\sigma, g_2) = e(H(T), \mathsf{vk})$. The BLS signature scheme is closely related to the identity-based encryption scheme of Boneh and Franklin [9]. Specifically, in the IBE scheme of [9] BLS verification keys take the role of the master public key, the signing key takes the role of the master secret key and signatures take the role of identity secret keys, where the signed messages correspond to the identities, respectively. In this sense, the BF scheme can be seen as a witness encryption scheme that allows to encrypt plaintexts m with respect to a verification key vk and a message T, such that anyone in possession of a valid signature of T under vk will be able to decrypt the plaintext m. Specifically, we can encrypt a message $m \in \{0,1\}$ by computing $\mathsf{ct} = (g_2^r, e(H(T), \mathsf{vk})^r \cdot g_T^m)$. Given a signature $\sigma = H(T)^x$, we can decrypt a ciphertext $\mathsf{ct} = (c_1, c_2)$ by computing $d = c_2/e(\sigma, c_1)$ and taking the discrete logarithm of d with respect to g_T (which can be done efficiently as $m \in \{0,1\}$).

SWE for BLS Multi-signatures The BLS scheme can be instantiated as an aggregatable multi-signature scheme [10]. Specifically, assume that for $i = 1, \ldots, n$ we have messages T_i with a corresponding signature σ_i with respect to a verification key vk_i. Then we can combine the signatures $\sigma_1, \ldots, \sigma_n$ into a *single* compact aggregate signature $\sigma = \prod_{i=1}^n \sigma_i$. Verifying such a signature can be done by checking whether $e(\sigma, g_2) = \prod_{i=1}^n e(H(T_i), \mathsf{vk}_i)$, where correctness follows routinely. We can adapt the BF IBE scheme to aggregate signatures in a natural way: To encrypt a plaintext $m \in \{0,1\}$ to messages $T_1, \ldots, T_n$ and corresponding verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ compute a ciphertext ct via $\mathsf{ct} = (g_2^r, (\prod_{i=1}^n e(H(T_i), \mathsf{vk}_i))^r \cdot g_T^m)$. Such a ciphertext $\mathsf{ct} = (c_1, c_2)$ can be decrypted analogously to the above by computing $d = c_2/e(\sigma, c_1)$ and taking the discrete logarithm with respect to g_T. To decrypt ct we need an aggregate signature σ of *all* T_i under their respective verification keys vk_i. For our envisioned applications this requirement is too strong, instead, we need a *threshold* scheme where a t-out-of-n aggregate signature suffices as a witness to decrypt a ciphertext. Thus, we will rely on Shamir's secret sharing scheme [38] to implement a t-out-of-n access structure. This, however, leads to additional challenges. Recall that Shamir's secret sharing scheme allows us to share a message $r_0 \in \mathbb{Z}_p$ into shares $s_1, \ldots, s_n \in \mathbb{Z}_p$, such that r_0 can be reconstructed via a (public) linear combination of any t of the s_i, while on the other hand, any set of less than t shares s_i reveals no information about r_0. The coefficients L_{i_j} of the linear combination required to reconstruct r_0 from a set of shares $s_{i_1}, \ldots, s_{i_t}$ (for indices $i_1, \ldots, i_t$) can be obtained from a corresponding set of Lagrange polynomials. Given such L_{i_j}, we can express r_0 as $r_0 = \sum_{j=1}^t L_{i_j} s_{i_j}$. We can now modify the above SWE scheme for aggregate signatures as follows. To encrypt a plaintext $m \in \{0,1\}$, we first compute a t-out-of-n secret sharing $s_1, \ldots, s_n$ of the plaintext m. The ciphertext ct is then computed by $\mathsf{ct} = (g_2^r, (e(H(T_i), \mathsf{vk}_i)^r \cdot g_T^{s_i})_{i \in [n]})$. Security of this scheme can be established from the same assumption as the BF IBE scheme, namely from the bilinear Diffie-Hellman (BDH) assumption [31]. We would now like to be able to decrypt such a ciphertext using an aggregate signature. For this purpose, however, we will have to modify the aggregation procedure of the aggregatable multi-signature scheme. Say we obtain t-out-of-n signatures σ_{i_j}, where σ_{i_j} is a signature of T_{i_j} under vk_{i_j}. Let L_{i_j} be the corresponding Lagrange coefficients. Our new aggregation procedure computes $\sigma = \prod_{j=1}^t \sigma_{i_j}^{L_{i_j}}$. That is, instead of merely taking the product of the σ_{i_j} we need to raise each σ_{i_j} to the power of its corresponding Lagrange coefficient L_{i_j}.

We can show that this modification does not hurt the security of the underlying aggregatable BLS multi-signature scheme. To decrypt a ciphertext $\mathsf{ct} = (c_0, c_1, \ldots, c_n)$ using such an aggregate signature σ, we compute $d = \prod_{j=1}^{t} c_{i_j}^{L_{i_j}} / e(\sigma, c_0)$ and take the discrete logarithm of d with respect to g_T. Correctness follows via a routine calculation.

Moving to the Source Group While the above scheme provides our desired functionality, implementing this scheme leads to a very poor performance profile. There are two main reasons: (1) Each ciphertext encrypts just a single bit. Thus, to encrypt any meaningful number of bits we need to provide a large number of ciphertexts. Observe that each ciphertext contains more than n group elements. Thus, encrypting k bits would require a ciphertext comprising kn group elements, which would be prohibitively large even for moderate values of k and n. (2) Both encryption and decryption rely heavily on pairing operations and operations in the target group. From an implementation perspective, pairing operations and operations in the target group are typically several times slower than operations in one of the source groups.

To address these issues, we will design a scheme that both allows for *ciphertext packing* and shifts almost all group operations into one of the two source groups (in our case this will be $\mathbb{G}_2$). This scheme is provided in Section 6.1 and we will only highlight a few aspects here.

- Instead of computing a secret sharing of the plaintext m, we compute a secret sharing of a random value $r_0 \in \mathbb{Z}_p$. The value r_0 can be used to randomize many batch-ciphertext components, leading to ciphertexts comprising only $O(k+n)$ group elements.
- We encrypt each share s_i in the source group $\mathbb{G}_2$ instead of $\mathbb{G}_T$. That is, we compute the ciphertext-component c_i via $c_i = \mathsf{vk}_i^r \cdot g_2^{s_i}$. This necessitates a corresponding modification of the decryption algorithm and requires that all messages T_i are identical, but this requirement is compatible with our envisioned applications. Somewhat surprisingly, this modification does not necessitate making a stronger hardness assumption, but only requires a rather intricate random-self-reduction procedure in the security proof. That is, even with this modification we can still rely on the hardness of the standard BDH assumption.
- Instead of just encrypting single bits $m \in \{0,1\}$, we allow the message m to come from $\{0, \ldots, 2^k - 1\}$. This will allow us to pack k bits into each ciphertext component. Recall that decryption requires the computation of a discrete logarithm with respect to a generator g_T. We can speed up this computation by relying on the Baby-Step-Giant-Step (BSGS) algorithm [40] to $O(2^{k/2})$ group operations. This leads to a very efficient implementation as the required discrete logarithm table for the fixed generator g_T can be precomputed.

A Compatibility-Layer for Efficient Proof Systems Our scheme so far assumes that encryptors behave honestly, i.e. the ciphertext ct is well-formed. A malicious encryptor, however, may provide ciphertexts that do not decrypt consistently, i.e. the decrypted plaintext m may depend on the signature σ used for decryption. Furthermore, for several of the use cases, we envision it is crucial to ensure that the encrypted message m satisfies additional properties. To facilitate this, we provide the following augmentations.

- We provide an efficient NIZK proof[5] in the ROM which ensures that ciphertexts decrypt consistently, i.e. the result of decryption does not depend on the signature which is used for decryption. This is provided in Section 7.1.

[5] Technically speaking, since our systems are only computationally sound, we provide non-interactive argument systems. However, to stay in line with the terminology of [26, 13] we refer to them as proof systems.

- We augment ciphertexts with efficient *proof-system enabled commitments* and provide very efficient plaintext equality proofs in the ROM. In essence, we provide an efficient NIZK proof system that allows to prove that a ciphertext ct and a Pedersen commitment C commit to the same value. This is discussed in Section 7.2.
- We can now rely on efficient and succinct proof systems such as Bulletproofs [13] to establish additional guarantees about the encrypted plaintext. For instance, we can rely on the range-proofs of [13] to ensure that the encrypted messages are within a certain range to ensure that our BSGS decryption procedure will recover the correct plaintext. This is discussed in Section 7.3.

To make this construction efficient, we include additional homomorphic commitments into SWE ciphertexts.

1.3 Related Work

Timed-release Crypto and "Encryption to the Future" The notion of timed-release encryption was proposed in the seminal paper by Rivest, Shamir and Wagner [37]. The goal is to encrypt a message so that it cannot be decrypted, not even by the sender, until a pre-determined amount of time has passed. This allows to "encrypt messages to the future". In [37] the authors propose two orthogonal directions for realizing such a primitive. Using trusted third-parties to hold the secrets and only reveal them once the pre-determined amount of time has passed, or by using so-called time-lock puzzles, that are computational problems that can not be solved without running a computer continuously for at least a certain amount of time.

An interesting example of the latter are timed commitments [12], which are commitments with an additional forced opening phase that requires a specified (big) amount of computation time. This is useful in an optimistic setting, where cooperation is usually the case, as an honest party can convince the receiver of the comitted value without needing to do the timely decryption step. This is indeed also possible for our SWE scheme, as we discuss in Section 7, that ciphertexts constitute a statistically binding commitment, but that is not our focus, as our decryption is efficient enough to be run. In case of one party aborting, timed commitments share all drawbacks of time-lock-puzzles, whereas our protocol works efficiently, even if the encryptor only submits their value and then goes offline.

Our approach is more closely related to the paradigm of using a trusted party as in [20, 18]. Simply put, these approaches set up a dedicated server that outputs tokens for decryption at specified times. We could deploy SWE in such a scenario as well, with the tokens being aggregated signatures on predictable messages. Specifically both [18] and our scheme achieve that no communication needs to take place between the trusted server and other entities. However, complete trust in a single (or multiple) server is a strong assumption, thus we re-use the decentralized architecture, computation and trust structure already present in blockchains.

With the advent of blockchains, multiple proposals to realize timed-release encryption using the blockchain as a time-keeping tool emerged, already. These previous results, presented here, are all more of theoretical interest, while we demonstrate practical efficiency of our scheme by our implementation reported in Section 8.

In [33] the authors propose a scheme based on extractable witness encryption using the blockchain as a reference clock; messages are encrypted to future blocks of the chain that once created can be used as a witness for decryption. However, extractable witness encryption is a very expensive primitive. Concurrently to this work, [17] proposes an "encryption to the future" scheme based on proof-of-stake blockchains. Their approach is geared at transmitting messages from past committee members to future slot winners of the proof-of-stake lottery and requires active participation in the protocol by the committee members. Our results differ from this by enabling encrypting to the future even for encryptors and decryptors that only read the state of the blockchain and we require no active participation of the committee beyond their regular duties, assuming, that predictable messages like a block header are already signed in each (finalized) block. Otherwise, all committees need to only include this one additional signature, irrespective of pending timed-encryptions, so there is no direct involvement between users of McFly and committees.

Another related line of work is presented in [6], where a message is kept secret and "alive" on the chain by re-sharing a secret sharing of the message from committee to committee. This allows to keep the message secret until an arbitrary condition is met and the committee can reveal the message. A more general approach is the recent YOSO protocol [29] that allows to perform secure computation in that same setting, by using an additive homomorphic encryption scheme, to which committees hold shares of a secret key and continuously re-share it. While these approaches realize some form of encryption to the future, they require massive communication from parties and are still far from practical.

A spin on timed commitments is also available using blockchains; in [1], a blockchain contract is introduced, that locks assets of the commitment sender for a set time based on a commitment. If the sender fails to open the commitment within that time, their assets are made available to the receiver as a penalty - however the commitment is not opened in that case.

Concurrent work: Between the publication of the initial version of this paper and this updated version, others have also worked on the interesting topic of timed-release encryption based on blockchains. We'd like to highlight two papers, that share similar constructions/techniques and can be considered concurrent. Both of them also rely heavily on the BLS signature and the Boneh-Franklin IBE scheme conceptually. In [27], a better scalability into the future is achieved by not running this system with respect to a generic blockchain, but by running it with the league of entropy in mind, which also publishes BLS signatures on predictable messages, but has the advantage that the public key of the committee remains constant. In [19], similar techniques to ours are used, but the focus is more on key derivation in the future with the help of third parties, whereas our paper focusses on sending messages into the future without third parties.

BLS Signatures and Identity-Based Encryption (IBE) The BLS signature scheme, introduced in [11], is a pairing-based signature scheme with signatures of one group element in size. Additionally, it is possible to aggregate signatures of multiple users on different messages, thus saving space as shown in [10]. Due to the very space-efficient aggregation, BLS signatures are used in widely deployed systems such as Ethereum 2.0 [24]. Aggregation for potential duplicate messages is achieved in [8, 36]. The former only allows to aggregate once, so all signatures have to be combined in one step; the latter uses proofs-of-possessions, where users need to show that they know the secret key corresponding to their public key to a key registration authority.

Identity based encryption was first introduced by Shamir [39]. The initial idea was to use the identity - e.g. a mailing address - as a public key that messages can be encrypted to. In a sense, our scheme can be seen as a threshold IBE, as we encrypt with respect to a committee and can only decrypt if a threshold of the committee members collaborate.

1.4 Organisation

We will provide definitions for all the cryptographic building blocks required in our work in Section 2 and give a definition of signature based witness encryption (SWE) in Section 3. We explain our McFly protocol, which integrates SWE with a blockchain and its properties and possible applications in Section 4. Then we proceed to construct the underlying primitives: an aggregatable multi-signature scheme based on BLS (Section 5), an SWE scheme (Section 6) and a proof-system to show well-formedness and additional properties of encrypted messages (Section 7). We conclude by giving an analysis of a prototype implementation of our SWE scheme in Section 8.

2 Preliminaries

We denote by $\lambda \in \mathbb{N}$ the security parameter and by $x \leftarrow \mathcal{A}(\mathsf{in}; r)$ the output of the algorithm $\mathcal{A}$ on input in where $\mathcal{A}$ is randomized with $r \leftarrow \{0,1\}^*$ as its randomness. We omit this randomness when it is obvious or not explicitly required. By $\mathcal{A}^O$ we denote, that we run $\mathcal{A}$ with oracle access to O, that is it may query the oracle on inputs of its choice and only receives the corresponding outputs. We denote by $x \leftarrow_\$ S$ an output x being chosen uniformly at random from a set S. We denote the set $\{1, \ldots, n\}$ by $[n]$. For a group element g we denote by $\langle g \rangle$ a canonic encoding of g as a bitstring. PPT denotes probabilistic polynomial time. Also, $poly(x), negl(x)$ respectively denote any polynomial or negligible function in parameter x.

Next, we define the cryptographic building blocks necessary for our protocol.

Aggregatable Multi-Signatures We need aggregatable signature schemes for which we can extract the secret keys. Aggregatable multi-signatures are digital signatures that allow to compress multiple signatures by multiple users on multiple messages that may contain duplicates into one aggregate signature. We require all published public keys to come with an online-extractable proof of knowledge that shows, that the issuer knows a corresponding secret key. This is similar to the multi-signature based on BLS constructed in [36] requiring all public keys to be registered with a key registry via a proof of possession; this proof can either be appended to one's public verification key or registered with a certifying authority/key registry and allows us to extract secret keys in our proofs. All algorithms below implicitly have oracle access to a hash function H as input.

Definition 1 (Aggregatable Multi-Signatures)**.** An aggregatable signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Agg}, \mathsf{AggVrfy}, \mathsf{Prove}, \mathsf{Valid})$ is a tuple of seven algorithms where:

- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$: The key generation algorithm takes a security parameter and outputs a pair of verification and signing keys $(\mathsf{vk}, \mathsf{sk})$.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, T)$: The signing algorithm takes as input a signing key sk and a message T. It outputs a signature σ.
- $b \leftarrow \mathsf{Vrfy}(\mathsf{vk}, T, \sigma)$: The verification algorithm takes as input a verification key vk, a message T and a signature σ. It outputs a bit b.
- $\sigma \leftarrow \mathsf{Agg}((\sigma_1, \ldots, \sigma_k), (\mathsf{vk}_1, \ldots, \mathsf{vk}_k))$: The aggregation algorithm takes a list of signatures $(\sigma_1, \ldots, \sigma_k)$ and verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$. It outputs one aggregate signature σ.
- $b \leftarrow \mathsf{AggVrfy}(\sigma, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k), (T_1, \ldots, T_k))$: The verification algorithm for aggregate signatures takes an aggregate signature σ as well as two lists of public verification keys vk_i and messages T_i, which may include duplicates. It outputs a bit b. For convenience, we consider this identical to Vrfy, if only σ and one key vk_1 and message T_1 are input.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{vk}, \mathsf{sk})$: The proving algorithm has access to an additional hash function oracle H_{pr}, takes a verification key vk and a signing key sk. It outputs a proof π.
- $b \leftarrow \mathsf{Valid}(\mathsf{vk}, \pi)$: The validity algorithm has access to to an additional hash function oracle H_{pr}, takes a verification key vk and a proof π. It outputs a bit b.

We require that such a signature scheme is correct and unforgeable and that $(\mathsf{Prove}, \mathsf{Valid})$ is an online-extractable zero-knowledge proof of knowledge for the key relation $\mathcal{K} = \{(\mathsf{vk}, \mathsf{sk}) | \exists r$ s.t. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda; r)\}$[6]

Definition 2 (Correctness)**.** An aggregatable multi-signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Agg}, \mathsf{AggVrfy}, \mathsf{Prove}, \mathsf{Valid})$ is *correct* if for all $\lambda \in \mathbb{N}$, $k = poly(\lambda)$, all messages $T, T_1, \ldots, T_k$, sets of public keys $V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$ and signatures $(\sigma_1, \ldots, \sigma_k)$ such that $\mathsf{Vrfy}(\mathsf{vk}_i, T_i, \sigma_i) = 1$ for $i \in [k]$ it holds:

$$\Pr\left[\begin{array}{l}\mathsf{Vrfy}(\mathsf{vk}, T, \mathsf{Sign}(\mathsf{sk}, T)) = 1 : \\ (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)\end{array}\right] = 1$$

and

$$\Pr\left[\begin{array}{l}\mathsf{AggVrfy}(\sigma, V, (T_1, \ldots, T_k)) = 1 : \\ \sigma \leftarrow \mathsf{Agg}((\sigma_1, \ldots, \sigma_k), V)\end{array}\right] = 1.$$

Definition 3 (Unforgeability)**.** An aggregatable multi-signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Agg}, \mathsf{AggVrfy}, \mathsf{Prove}, \mathsf{Valid})$ is *unforgeable* if for all $\lambda \in \mathbb{N}$, $N = poly(\lambda)$ there is no PPT adversary $\mathcal{A}$ with more than negligible advantage $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Unf}} = \Pr[\mathsf{Exp}_{\mathsf{Unf}}(\mathcal{A}, N) = 1]$ in the experiment $\mathsf{Exp}_{\mathsf{Unf}}(\mathcal{A}, N)$.

Experiment $\mathsf{Exp}_{\mathsf{Unf}}(\mathcal{A}, N)$

- The experiment randomly chooses $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and outputs vk to $\mathcal{A}$.
- $\mathcal{A}$ may request signatures from a signing oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$.
- $\mathcal{A}$ may also request to see a proof for vk, which the experiment answers with $\pi \leftarrow \mathsf{Prove}(\mathsf{vk}, \mathsf{sk})$. If it does, $\mathcal{A}$ may not use vk as one of the keys in its forge.
- Eventually, $\mathcal{A}$ outputs $((T_1, \ldots, T_k), (\mathsf{vk}_2, \ldots, \mathsf{vk}_k), (\pi_2, \ldots, \pi_k), \sigma^*)$ as a forge. If $\mathcal{A}$ requested a proof for vk before, but one of the vk_i for $i \in \{2, \ldots, k\}$ is vk, the experiment outputs 0. The case $k = 1$, where $\mathcal{A}$ outputs no additional keys is explicitly allowed.
- The experiment outputs 1, if $\mathsf{AggVrfy}(\sigma^*, (\mathsf{vk}, \mathsf{vk}_2, \ldots, \mathsf{vk}_k), (T_1, \ldots, T_k)) = 1$, T_1 was never queried to the oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$, $\mathsf{Valid}(\mathsf{vk}_i, \pi_i) = 1$ for all $i \in \{2, \ldots, k\}$ and the number of keys k is upper bounded by N. Otherwise, the output is 0.

Hash Functions A keyed family of hash functions $\mathbb{H}$ consists of the following functions:

- $k \leftarrow \mathsf{KeyGen}(1^\lambda)$: The key generation algorithm takes a security parameter and outputs a key k.
- $h \leftarrow \mathbb{H}_k(m)$: The hash algorithm takes as input a key k and a message m. It outputs a hash h.

We expect a hash function family to be collision resistant.

Definition 4 (Collision Resistance)**.** A family of hash functions is collision resistant, if for any PPT adversary $\mathcal{A}$

$$\Pr\left[\begin{array}{l}m_1 \neq m_2 \text{ and } H_k(m_1) = H_k(m_2) : \\ k \leftarrow \mathsf{KeyGen}(1^\lambda); (m_1, m_2) \leftarrow \mathcal{A}(k)\end{array}\right] < negl(\lambda)$$

[6]Note, that this requires implicitly, that it is efficiently checkable, whether a secret key belongs to a given public key. This will be true for our use-case.

In the following, for convenience we will omit the key and assume it has been handed out as a public hash function $H = H_k$.

Furthermore, we will be using the random oracle model, in which a hash function can only be evaluated by directly querying the hashing oracle [5]. The output on a message that has not been queried yet is uniform and determined at the time of the query. This is a heuristic for the behaviour of hash functions that allows us to simulate the hashing oracle ourselves in our reductions.

Pseudo-Random Functions A keyed family of functions $\mathsf{PRF}_k : \{0,1\}^s \to \{0,1\}^t$ for keys $k \in \{0,1\}^*$ and some $s, t = poly(|k|)$ is a pseudo-random function (PRF) family, if

- given k, m the function $\mathsf{PRF}_k(m)$ is efficiently computable and
- for every PPT distinguisher $\mathcal{D}$, it holds $\mathcal{D}^{\mathsf{PRF}_k(\cdot)} \approx_c \mathcal{D}^{F(\cdot)}$, where $k \leftarrow_\$ \{0,1\}^\lambda$ and F is chosen randomly from all functions from $\{0,1\}^{s(\lambda)}$ to $\{0,1\}^{t(\lambda)}$.

Commitment Schemes A (non-interactive) commitment scheme (CS) $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Vrfy})$ is composed of the following algorithms:

- $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$: The setup algorithm takes the security parameter λ and outputs a common reference string CRS.
- $(\mathsf{com}, \gamma) \leftarrow \mathsf{Commit}(\mathsf{CRS}, m)$: The commit algorithm takes as input a common reference string CRS and a message m. It outputs a commitment com and opening information γ.
- $b \leftarrow \mathsf{Verify}(\mathsf{CRS}, \mathsf{com}, m, \gamma)$: The verification algorithm takes as input a common reference string CRS, a commitment com, a message m and opening information γ. It outputs a bit $b \in \{0,1\}$.

Definition 5 (Correctness)**.** We say that a commitment scheme is correct if for all $\lambda \in \mathbb{N}$, $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$ and every message m we have that

$$\Pr\left[1 \leftarrow \mathsf{Verify}(\mathsf{CRS}, \mathsf{com}, m, \gamma) : (\mathsf{com}, \gamma) \leftarrow \mathsf{Commit}(\mathsf{CRS}, m)\right] = 1.$$

Definition 6 (Computational Hiding)**.** We say that a commitment scheme is computationally hiding if for all $\lambda \in \mathbb{N}$, $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$ and all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have that :

$$\left|\Pr\left[\begin{array}{ccc} & & (m_0, m_1, aux) \leftarrow \mathcal{A}_1(\mathsf{CRS}) \\ b = b' & : & b \leftarrow_\$ \{0,1\} \\ & & (\mathsf{com}, \gamma) \leftarrow \mathsf{Commit}(\mathsf{CRS}, m_b) \\ & & b' \leftarrow \mathcal{A}_2(\mathsf{com}, aux) \end{array}\right] - \frac{1}{2}\right| = negl(\lambda)$$

Definition 7 (Perfect Binding)**.** We say that a commitment scheme is perfectly binding if for all $\lambda \in \mathbb{N}$, $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$ and all adversaries $\mathcal{A}$ we have that

$$\Pr\left[\begin{array}{cc} & (\mathsf{com}, m_0, \gamma_0, m_1, \gamma_1) \leftarrow \mathcal{A}(\mathsf{CRS}) \\ m_0 \neq m_1 \wedge b = b' = 1 : & b \leftarrow \mathsf{Verify}(\mathsf{CRS}, \mathsf{com}, m_0, \gamma_0) \\ & b' \leftarrow \mathsf{Verify}(\mathsf{CRS}, \mathsf{com}, m_1, \gamma_1) \end{array}\right] = 0.$$

Zero-Knowledge Proofs We require two types of proof systems in the random oracle model which we will describe below. Definitions are partially taken from [26]:

Definition 8 (Zero-Knowledge Proof of Knowledge)**.** A proof of knowledge for an NP language $\mathcal{L}$ defined by an efficiently verifiable binary relation $\mathcal{R}$ via $x \in \mathcal{L} \Leftrightarrow \exists w$ s.t. $(x, w) \in \mathcal{R}$ consists of the following functions, which have access to a hash function H:

- $\pi \leftarrow \mathsf{Prove}^H(x, w)$: The proof algorithm takes a statement x and a witness w. It outputs a proof π.
- $b \leftarrow \mathsf{Vrfy}^H(x, \pi)$: The verification algorithm takes as input a statement x and a proof π. It outputs a bit b.

A *zero-knowledge proof of knowledge* in the random oracle model consist of $(\mathsf{Prove}, \mathsf{Vrfy})$ fulfilling completeness, zero-knowledge and the proof-of-knowledge property as below. An *online-extractable* zero-knowledge proof of knowledge fulfills the stronger assumption of extractability instead of the proof-of-knowledge property:

Definition 9 (Completeness)**.** A proof system $(\mathsf{Prove}, \mathsf{Vrfy})$ is complete, if for any $(x, w) \in \mathcal{R}$, any hash function H, it holds

$$\Pr\left[\mathsf{Vrfy}^H(x, \mathsf{Prove}^H(x, w)) = 1\right] = 1$$

Definition 10 (Zero-Knowledge)**.** We say that a proof system $(\mathsf{Prove}, \mathsf{Vrfy})$ is zero-knowledge in the random oracle model, if there exists a PPT simulator S such that for all PPT distinguishers $\mathcal{D}$ the following distributions are computationally indistinguishable:

- Let H be a random oracle, set $\pi_0 = \emptyset$, $\delta_0 = 1^\lambda$. Repeat for $i = 1, \ldots, n$ until $\mathcal{D}$ stops: $(x_i, w_i, \delta_i) \leftarrow \mathcal{D}^H(i, \pi_{i-1}, \delta_{i-1})$, where $\pi_i \leftarrow \mathsf{Prove}^H(x_i, w_i)$ if $(x_i, w_i) \in \mathcal{R}$ or $\pi \leftarrow \perp$ otherwise. Output $\mathcal{D}$'s final output.
- Let $(H_0, \tau_0) \leftarrow S(0, 1^\lambda)$, set $\pi_0 = \emptyset$, $\delta_0 = 1^\lambda$. Repeat for $i = 1, \ldots, n$ until $\mathcal{D}$ stops: $(x_i, w_i, \delta_i) \leftarrow \mathcal{D}^{H_{i-1}}(i, \pi_{i-1}, \delta_{i-1})$, where $(H_i, \pi_i, \tau_i) \leftarrow S(i, x_i, \tau_{i-1}, \mathtt{YES})$ if $(x_, w_i) \in \mathcal{R}$ or $(H_i, \pi_i, \tau_i) \leftarrow S(i, x_i, \tau_{i-1}, \mathtt{NO})$ otherwise. Output $\mathcal{D}$'s final output.

Definition 11 (Proof of Knowledge)**.** We say that a proof system $(\mathsf{Prove}, \mathsf{Vrfy})$ is a proof of knowledge with knowledge error ε, if there exists a PPT extractor $\mathcal{E}$, such that for every PPT prover P^* and input x it holds:

$$Pr\left[(x, w) \notin \mathcal{R} : w \leftarrow \mathcal{E}^{\mathsf{P}^*}(x)\right] \leq Pr\left[\mathsf{Vrfy}(x, \pi) = 0 : \pi \leftarrow \mathsf{P}^*(x)\right] - \varepsilon$$

Here $\mathcal{E}^{\mathsf{P}^*}$ denotes that the extractor gets full black-box access to the algorithm P^* including the power to rewind.

Definition 12 (Extractability)**.** There exists a PPT extractor $\mathcal{E}$, such that for every PPT algorithm $\mathcal{A}$ and the simulator S from the zero-knowledge definition, it holds:
Let $(H_0, \tau_0) \leftarrow S(0, 1^\lambda)$, set $\pi_0 = \emptyset$, $\delta_0 = 1^\lambda$. Repeat for $i = 1, \ldots, n$ until $\mathcal{A}$ stops: $(x_i, w_i, \delta_i) \leftarrow \mathcal{A}^{H_{i-1}}(i, \pi_{i-1}, \delta_{i-1})$, where $(H_i, \pi_i, \tau_i) \leftarrow S(i, x_i, \tau_{i-1}, \mathtt{YES})$ if $(x_, w_i) \in \mathcal{R}$ or $(H_i, \pi_i, \tau_i) \leftarrow S(i, x_i, \tau_{i-1}, \mathtt{NO})$ otherwise. Let (x, π) be $\mathcal{A}$'s final output and and $Q_\mathcal{A}$ be the queries that $\mathcal{A}$ made to oracles H_i. Let $w \leftarrow \mathcal{E}(x, \pi, Q_\mathcal{A})$. Then, if $(x, \pi) \neq (x_i, \pi_i)$ for all $i \in [n]$,

$$Pr\left[(x, w) \notin \mathcal{R} \wedge \mathsf{Vrfy}^{H_n}(\mathsf{vk}, \pi) = 1\right] \leq negl(\lambda)$$

Extractability essentially follows the same idea as proof-of-knowledge, but additionally doesn't allow rewinding and was written in the multi-statement model, because we need these stronger guarantees for part of our construction.

Definition 13 (Non-interactive Zero-Knowledge Proof Systems)**.** A *NIZK proof system* for an NP language $\mathcal{L}$ defined by an efficiently verifiable binary relation $\mathcal{R}$ via $x \in \mathcal{L} \Leftrightarrow \exists w$ s.t. $(x, w) \in \mathcal{R}$ consists of $(\mathsf{Prove}, \mathsf{Vrfy})$ as above and fulfills completeness, zero-knowledge and computational soundness[7]:

[7] As stated above, this is typically referred to as an argument system, but we call it a proof for consistency with prior works.

Definition 14 (Computational Soundness)**.** We say that a proof system $(\mathsf{Prove}, \mathsf{Vrfy})$ has computational soundness if for all $\lambda \in \mathbb{N}$, every collision resistant hash function H, every $x \notin \mathcal{L}$ and PPT adversaries $\mathcal{A}$, it holds

$$\Pr\left[\ \mathsf{Vrfy}^H(x, \pi) = 1 : \pi \leftarrow \mathcal{A}^H(1^\lambda, x)\ \right] = negl(\lambda)$$

Secret Sharing and Coding Theory We will briefly introduce some elementary concepts relating to Shamir's secret sharing and its underlying coding structure, Reed-Solomon codes. Let $\mathbb{Z}_p$ be the finite field of prime order p and fix distinct elements $\xi = \xi_1, \ldots, \xi_n \in \mathbb{Z}_p$. The Reed-Solomon code $RS_{n,k}[\xi]$ consists of all vectors $\mathbf{c} = (f(\xi_1), \ldots, f(\xi_n))$ for some polynomial $f(X) = \sum_{i=0}^{k-1} a_i X^i$ of degree $k-1$. This code is generated by the matrix $\mathbf{G} = (\xi_i^j)_{i,j} \in \mathbb{Z}_p^{n \times k}$ and has a parity-check matrix $\mathbf{H} = \left(\frac{1}{\prod_{l \neq j}(\xi_j - \xi_l)} \xi_j^i\right)_{i,j} \in \mathbb{Z}_p^{(n-k) \times n}$.

Lagrange Interpolation For a set of supporting points $\chi_1, \ldots, \chi_k$ from a finite field $\mathbb{Z}_p$, where $p \in \mathbb{N}$ is prime, the Lagrange basis polynomials are given by $L_1, \ldots, L_k$, where

$$L_i(x) = \prod_{j \in [k]; j \neq i} \frac{x - \chi_j}{\chi_i - \chi_j}.$$

These are chosen such that $L_i(\chi_j) = 1$ iff $i = j$ and 0 otherwise. Consequently, given a set of k data points (χ_i, y_i), we can output a polynomial $f_L(x) = \Sigma_{i \in [k]} L_i(x) \cdot y_i$ that will run through these points and which has degree at most k-1. This process is called Lagrange Interpolation.

Bilinear Group Setting We regard the same setup as in [10], that is we assume groups G_1, G_2, G_T of prime order p with their respective generators g_1, g_2 and g_T. Additionally, we assume a computable bilinear map $e : G_1 \times G_2 \to G_T$. That is, e has the following properties:

1. Bilinearity: for all $u \in G_1, v \in G_2$ and $a, b \in \mathbb{Z}$, it holds that $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-Degeneration: $e(g_1, g_2) \neq 1$.

From these, it follows also that for any $u_1, u_2 \in G_1, v \in G_2$ it holds $e(u_1 \cdot u_2, v) = e(u_1, v) \cdot e(u_2, v)$. We assume that the group operations in all of these groups as well as e can be computed in one time step and that the computational Co-Diffie-Hellman assumption holds in (G_1, G_2) and the bilinear Diffie-Hellman assumption hold in (G_1, G_2, G_T). In some instances we additionally require the knowledge of exponent assumption to hold in (G_1, G_2, G_T).

Definition 15 (Computational Co-Diffie-Hellman)**.** The computational Co-Diffie-Hellman assumption for a pair of groups (G_1, G_2) states that the probability

$$Pr\left[A(g_1, g_1^x, g_2, g_2^x, h) = h^x : x \leftarrow_\$ \mathbb{Z}_p, h \leftarrow_\$ G_1\right]$$

is negligible for polynomial adversaries $\mathcal{A}$, where $g_1 \in G_1, g_2 \in G_2$ are generators.

Definition 16 (Bilinear Diffie-Hellman)**.** The bilinear Diffie-Hellman assumption for a triple of groups (G_1, G_2, G_T) of order p states that the following distributions are computationally close:

$$(g_1, g_1^x, g_1^\alpha, g_2, g_2^x, g_2^r, g_T^{\alpha x r}) \approx_c (g_1, g_1^x, g_1^\alpha, g_2, g_2^x, g_2^r, g_T^y),$$

where $x, y, \alpha, r \leftarrow_\$ \mathbb{Z}_p$ and $g_1 \in G_1, g_2 \in G_2, g_T = e(g_1, g_2) \in G_T$ are generators.

We will now adapt the knowledge of exponent assumption [4] to the bilinear setting. Similar to the original one this assumption holds generically.

Definition 17 (Knowledge of Exponent Assumption)**.** For a pair of groups (G_1, G_2, G_T) with generators (g_1, g_2, g_T) of prime orders p as above this assumption states that if there exists a PPT adversary $\mathcal{A}$ where:

- $\mathcal{A}$ takes as input a generator $h \in G_1$.
- $\mathcal{A}$ outputs two group elements $C \in G_1, Y \in G_2$ such that $e(h, Y) = e(C, g_2)$, that is (g_2, Y) and (h, C) have the same dlog relationship.

If such an $\mathcal{A}$ exists, then there exists a PPT extractor $\bar{\mathcal{A}}$, that takes the same input (and potentially randomness) as $\mathcal{A}$ and outputs the same C, Y and additionally c, such that $C = h^c, Y = g_2^c$.

Expected-Time Forking Lemma We will use the following forking lemma proven secure in [3]:

Lemma 1 (Expected-Time Forking Lemma)**.** *Let H be a set of size $h \geq 2$, $q \geq 1$ and A a randomized algorithm, that on input x, h_1, h_2 returns an integer from $0, 1, 2$ in at most $T_A(|x|)$ time steps. Let B be a brute-force algorithm, that takes input x and halts after at most $T_B(|x|)$ steps, outputting* 1*.*

Let the accepting probability $P_A(x)$ be defined as the probability of the following experiment outputting 1*:*

Randomly choose coins ρ for A
Pick $h_1, \ldots, h_q \leftarrow_\$ H$ and set $i \leftarrow A(x, h_1, \ldots, h_q; \rho)$
If $i \geq 1$, return 1*, else return* 0*.*

Then it holds that the forking algorithm $F_A(x)$ given as:

Randomly choose coins ρ for A
Pick $h_1, \ldots, h_q \leftarrow_\$ H$ and set $i \leftarrow A(x, h_1, \ldots, h_q; \rho)$
If $i = 0$, return 0.
Repeat, in parallel with $B(x)$:
Pick $h'_i, \ldots, h'_q \leftarrow_\$ H$
Set $j \leftarrow A(x, h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_q; \rho)$
Until ($i = j$ and $h'_i \neq h_i$) or B has halted.
Return 1

returns 1 *with the same probability $P_A(x)$ in expected time $T_{F_A}(|x|) \leq (4q+1)T_A(|x|) + \frac{4q}{h} T_B(|x|)$.*

3 Signature-Based Witness Encryption

We now formally define the new cryptographic primitive SWE, which is the core technical component of our McFly protocol.

Definition 18 (Signature-Based Witness Encryption)**.** A t-out-of-n SWE for an aggregate signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Agg}, \mathsf{AggVrfy}, \mathsf{Prove}, \mathsf{Valid})$ is a tuple of two algorithms $(\mathsf{Enc}, \mathsf{Dec})$ where:

- $\mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n), (T_i)_{i \in [\ell]}, (m_i)_{i \in [\ell]})$: Encryption takes as input a set V of n verification keys of the underlying scheme Sig, a list of reference signing messages T_i and a list of messages m_i of arbitrary length $\ell \in poly(\lambda)$. It outputs a ciphertext ct.

- $m \leftarrow \mathsf{Dec}(\mathsf{ct}, (\sigma_i)_{i\in[\ell]}, U, V)$: Decryption takes as input a ciphertext ct, a list of aggregate signatures $(\sigma_i)_{i\in[\ell]}$ and two sets U, V of verification keys of the underlying scheme Sig. It outputs a message m.

In our construction, we explicitly restrict the message-space: for all i, $m_i \in \{0, 2^k - 1\}$ must hold for appropriate k. The reason is, that we will have to compute a dlog to retrieve the message and we choose a sensible bound to enable the baby-step-giant-step method to be efficient.

We require such a scheme to fulfill two properties: robust correctness and security. The idea is to model fine-grained access; When we encrypt messages m_i under reference messages T_i, then we can decrypt m_{ind} at a specific index ind iff we get an aggregated signature of T_{ind} under at least t keys for that index.

Definition 19 (Robust Correctness)**.** A t-out-of-n SWE scheme $\mathsf{SWE} = (\mathsf{Enc}, \mathsf{Dec})$ for an aggregate signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Agg}, \mathsf{AggVrfy}, \mathsf{Prove}, \mathsf{Valid})$ is *correct* if for all $\lambda \in \mathbb{N}$ and $\ell = poly(\lambda)$ there is no PPT adversary $\mathcal{A}$ with more than negligible probability of outputting an index $\mathsf{ind} \in [\ell]$, a set of keys $V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$, a subset $U \subseteq V$ with $|U| \geq t$, message lists $(m_i)_{i\in[\ell]}, (T_i)_{i\in[\ell]}$ and signatures $(\sigma_i)_{i\in[\ell]}$, such that $\mathsf{AggVrfy}(\sigma_{\mathsf{ind}}, U, (T_{\mathsf{ind}})_{i\in[|U|]}) = 1$, but $\mathsf{Dec}(\mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, (m_i)_{i\in[\ell]}), (\sigma_i)_{i\in[\ell]}, U, V)_{\mathsf{ind}} \neq m_{\mathsf{ind}}$.

Definition 20 (Security)**.** A t-out-of-n SWE scheme $\mathsf{SWE} = (\mathsf{Enc}, \mathsf{Dec})$ for an aggregate signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Agg}, \mathsf{AggVrfy}, \mathsf{Prove}, \mathsf{Valid})$ is *secure* if for all $\lambda \in \mathbb{N}$, such that $t = poly(\lambda)$, and all $\ell = poly(\lambda)$, subsets $SC \subseteq [\ell]$, there is no PPT adversary $\mathcal{A}$ that has more than negligible advantage in the experiment $\mathsf{Exp}_{\mathsf{Sec}}(\mathcal{A}, 1^\lambda)$. We define $\mathcal{A}$'s advantage by $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Sec}} = |\Pr\left[\mathsf{Exp}_{\mathsf{Sec}}(\mathcal{A}, 1^\lambda) = 1\right] - \frac{1}{2}|$.

Experiment $\mathsf{Exp}_{\mathsf{Sec}}(\mathcal{A}, 1^\lambda)$

1. Let H_{pr} be a fresh hash function from a keyed family of hash functions, available to the experiment and $\mathcal{A}$.
2. The experiment generates $n - t + 1$ key pairs for $i \in \{t, \ldots, n\}$ as $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig}.\mathsf{KeyGen}(1^\lambda)$ and provides vk_i as well as $\mathsf{Sig}.\mathsf{Prove}^{H_{pr}}(\mathsf{vk}_i, \mathsf{sk}_i)$ for $i \in \{t, \ldots, n\}$ to $\mathcal{A}$.
3. $\mathcal{A}$ inputs $VC = (\mathsf{vk}_1, \ldots, \mathsf{vk}_{t-1})$ and $(\pi_1, \ldots, \pi_{t-1})$. If for any $i \in [t-1]$, $\mathsf{Sig}.\mathsf{Valid}(\mathsf{vk}_i, \pi_i) = 0$, we abort. Else, we define $V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$.
4. $\mathcal{A}$ gets to make signing queries for pairs (i, T). If $i < t$, the experiment aborts, else it returns $\mathsf{Sig}.\mathsf{Sign}(\mathsf{sk}_i, T)$.
5. The adversary announces challenge messages m_i^0, m_i^1 for $i \in SC$, a list of messages $(m_i)_{i\in[\ell]\setminus SC}$ and a list of signing reference messages $(T_i)_{i\in[\ell]}$. If a signature for a T_i with $i \in SC$ was previously queried, we abort.
6. The experiment flips a bit $b \leftarrow_\$ \{0, 1\}$, sets $m_i = m_i^b$ for $i \in SC$ and sends $\mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, (m_i)_{i\in[\ell]})$ to $\mathcal{A}$.
7. $\mathcal{A}$ gets to make further signing queries for pairs (i, T). If $i \geq t$ and $T \neq T_i$ for all $i \in SC$, the experiment returns $\mathsf{Sig}.\mathsf{Sign}(\mathsf{sk}_i, T)$, else it aborts.
8. Finally, $\mathcal{A}$ outputs a guess b'.
9. If $b = b'$, the experiment outputs 1, else 0.

Definition 21 (Verifiable Signature-Based Witness Encryption)**.** A scheme $\mathsf{SWE} = (\mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Vrfy})$ is a verifiable SWE for relation $\mathcal{R}$, if $\mathsf{Enc}, \mathsf{Dec}$ are as above and $\mathsf{Prove}, \mathsf{Vrfy}$ are a NIZK proof system for a language given by the following induced relation $\mathcal{R}'$, where $V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ is a set of keys:

$$(V,(T_i)_{i\in[\ell]}, \mathsf{ct}), ((m_i)_{i\in[\ell]}, w, r)) \in \mathcal{R}' \Leftrightarrow$$
$$\mathsf{ct} = \mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, (m_i)_{i\in[\ell]}); r) \text{ and } (m = \sum_{i\in[\ell]} 2^{(i-1)k} m_i, w) \in \mathcal{R}$$

4 The McFly Protocol

In this section, we describe how to build a general-purpose time-release encryption mechanism, that we call McFly, by integrating a verifiable signature based witness encryption SWE with a blockchain. The time-release mechanism is available to all users of the underlying blockchain. This allows for countless complex applications with a minimal overhead. First we introduce a formal model for the underlying blockchain and then we give a concrete instantiation of the McFly protocol. Later, we discuss concrete applications that can take advantage of the time-release encryption mechanism provided by the combination of SWE and the blockchain.

4.1 Formal Model and Guarantees

In this section we introduce a simplified model for blockchains in the form of the $\mathcal{BC}_{\lambda,H}$ functionality reflecting the requirements introduced in Section 1.1.

As we are modelling BFT blockchains and blockchains coupled with a finality layer, all the blocks in our abstraction are final and cannot be rolled-back. Parties only require the signatures of the committee members on the block counter to decrypt ciphertexts, thus the blockchain in our model simply consists of a list T containing these signatures. On every tick, committee members sign the block and the new counter ctr as the block header. Our model takes a security parameter λ and two hash functions H, H_{pr} as parameters.

Let the number of committee members be n and the corruption threshold of the adversary be $c < n/2$. We allow the adversary to corrupt up to c parties statically - that is they may choose their signing keys in the beginning of the execution and input the signatures used in making the aggregated signatures on block counter for these parties later on.[8] We additionally require an online-extractable proof of knowledge for the public keys of committee members. Further, the adversary gets to decide when to make a new block (up to delay Δ_τ per round) by calling Tick, - we allow them full control over the content of these blocks, except for the fixed block counters. They also get to see all unagreggated signatures by the honest parties.

Functionality $\mathcal{BC}_{\lambda,H}$

Initialization

$\mathsf{T} := ()$
$ctr := 0$ ▷ Current number of blocks
$\mathcal{C} := \emptyset$ ▷ Set of corrupted parties
Wait until $(\mathsf{Corrupt}, \cdot)$ is called by adversary $\mathcal{A}$
for $i \in [n]$ **do**
  if $i \notin \mathcal{C}$ **then**

[8] We consider static corruptions for simplicity. However, we can easily extend our model to allow for *delayed* active corruptions. In such a model, the committees are dynamically sampled and the adversary is only allowed to corrupt parties after a specified delay; this delay can then be set to be larger than the time a new committee is sampled.

Sample $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Sig}'.\mathsf{KeyGen}(1^\lambda)$
Output $\pi_i \leftarrow \mathsf{Sig}'.\mathsf{Prove}(\mathsf{vk}_i, \mathsf{sk}_i)$ to $\mathcal{A}$.
$\mathcal{V} := \{\mathsf{vk}_i\}_{i\in[n]}$

Public Interface

Input: $(\mathsf{QueryAt}, CTR)$
if $CTR \leq \mathsf{T}.\mathsf{len}()$ **then**
return $\mathsf{T}[CTR]$

Input: $(\mathsf{QueryTime})$
return $\mathsf{T}.\mathsf{len}()$

Input: $(\mathsf{QueryKeys})$
return $\mathcal{V}$

Interface for Adversary $\mathcal{A}$

Input: $(\mathsf{Corrupt}, C', \{\mathsf{vk}'_i\}_{i\in C'}, \{\pi_i\}_{i\in C'})$ ▷ This can be called once during initialization, modelling static corruption
if $|C'| \leq c$ and $C' \subset [n]$ and $\mathsf{Sig}'.\mathsf{Valid}^{H_{pr}}(\mathsf{vk}'_i, \pi_i)$ for $i \in C'$ **then**
$\mathcal{C} := C'$
for $i \in \mathcal{C}$ **do**
Set $\mathsf{vk}_i = \mathsf{vk}'_i$

Input: (Tick, m)
$ctr+ = 1$
for $i \in ([n] \setminus \mathcal{C})$ **do**
Output $(\sigma_i) \leftarrow \mathsf{Sig}'.\mathsf{Sign}(\mathsf{sk}_i, H(ctr))$ to $\mathcal{A}$
Output $(\sigma'_i) \leftarrow \mathsf{Sig}'.\mathsf{Sign}(\mathsf{sk}_i, H(ctr, m))$ to $\mathcal{A}$
Set $S := [n] \setminus \mathcal{C}$.
Await **input** $(C', (\sigma_i)_{i\in C'})$ from $\mathcal{A}$
if for all $i \in C', \mathsf{Sig}'.\mathsf{Vrfy}(\mathsf{vk}_i, H(ctr), \sigma_i) = 1$ **then**
$S := S \cup C'$.
Append $(\mathsf{Sig}'.\mathsf{Agg}((\sigma_i)_{i\in S}, (\mathsf{vk}_i)_{i\in S}), (\mathsf{vk}_i)_{i\in S})$ to T.

Restrictions on the Adversary

For each round r_i, the adversary is required to send a message (Tick, m) for some block content m within time Δ_τ.

Protocol Guarantees Let $\mathcal{L}_0$ be an NP language defined by relation $\mathcal{R}_0$ via $m \in \mathcal{L}_0 \Leftrightarrow \exists w$ s.t. $(m, w) \in \mathcal{R}_0$. Our protocol McFly consists of five algorithms (Setup, Enc, Dec, Prove, Vrfy) in a hybrid model where access to the public interface of the functionality $\mathcal{BC} = \mathcal{BC}_{\lambda,H}$ is assumed with committee size $n = poly(\lambda)$ and corruption threshold $c < n/2$. The syntax of these algorithms is as follows:

$\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$: Setup takes a security parameter λ. It outputs a common reference string CRS.

$\mathsf{ct} \leftarrow \mathsf{Enc}^{\mathcal{BC}}(1^\lambda, m, d)$: Encryption takes a security parameter λ, a message m and an encryption depth d. It outputs a ciphertext ct.

$m \leftarrow \mathsf{Dec}^{\mathcal{BC}}(\mathsf{ct}, d)$: Decryption takes a ciphertext ct and an encryption depth d. It outputs a message m.

$\pi \leftarrow \mathsf{Prove}^{\mathcal{BC}}(1^\lambda, \mathsf{CRS}, \mathsf{ct}, m, d, w_0, r)$: The proving algorithm takes a security parameter λ, CRS, a message m, an encryption depth d, a witness w_0 and randomness r. It outputs a proof π.

$b \leftarrow \mathsf{Vrfy}^{\mathcal{BC}}(\mathsf{CRS}, \mathsf{ct}, \pi, d)$: The verification algorithm takes CRS, a ciphertext ct, a proof π and an encryption depth d. It outputs a bit b.

We prove the following security guarantees for McFly, which are inspired by traditional time-lock puzzles:

Definition 22 (Correctness)**.** A protocol $\mathsf{McFly} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Vrfy})$ is correct, if for any parameter λ, message m, depth d, and algorithm $\mathcal{A}$ running the adversarial interface in $\mathcal{BC}$, if $\mathsf{ct} \leftarrow \mathsf{Enc}^{\mathcal{BC}}(1^\lambda, m, d)$ is run at any point and $\mathsf{McFly.Dec}^{\mathcal{BC}}(\mathsf{ct}, d)$ is run, when the number of finalized blocks $\mathcal{BC}.\mathsf{QueryTime}$ is at least d, it will output m, except with negligible probability.

Definition 23 (Security)**.** A protocol $\mathsf{McFly} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{Dec})$ is secure, if for any parameter λ and committee size $n = poly(\lambda)$, corruption threshold $c < n/2$ there is no PPT adversary $\mathcal{A}$ with more than negligible advantage $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Lock}} = |\Pr[b = b'] - \frac{1}{2}|$ in the experiment $\mathsf{Exp}_{\mathsf{Lock}}(\mathcal{A}, 1^\lambda)$.

Experiment $\mathsf{Exp}_{\mathsf{Lock}}(\mathcal{A}, 1^\lambda)$

1. The experiment computes $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$ and outputs it to $\mathcal{A}$.
2. $\mathcal{A}$ gets to use the adversarial interface in $\mathcal{BC}$, which is run by the experiment.
3. At some point, $\mathcal{A}$ sends two challenge messages m_0, m_1 and a depth $d > 0$. $|m_0| = |m_1|$ must hold.
4. The experiment draws $b \leftarrow_\$ \{0, 1\}$.
5. Run $\mathsf{ct} \leftarrow \mathsf{Enc}^{\mathcal{BC}}(1^\lambda, m_b, d)$ and send ct to $\mathcal{A}$.
6. $\mathcal{A}$ can submit a bit b' at any point while $ctr < d$ in $\mathcal{BC}$
7. Once $ctr \geq d$ on $\mathcal{BC}$ with no prior input from $\mathcal{A}$, $b' \leftarrow_\$ \{0, 1\}$ is set instead.

Remark. There are two different points in time that are used in these guarantees - When ctr is incremented, can be seen as the point in time when it is clear that the committee has reached agreement, and that a new finalized block will be added. When the aggregated signature is added to T symbolizes the point in time when the finalized block (including the committee signatures) becomes available to all honest users on the blockchain (even outside of the committee). We point out that our synchronous network model guarantees that all honest parties receive the messages of other honest parties by the end of each round. However, the best practical guarantee that we can hope to achieve is that an adversary corrupting up to $n/2 - 1$ committee members is unable to decrypt a ciphertext before seeing any honest member's signature. However, this might occur before a block is made available to honest users. In practice, such a gap exists naturally, as we also need to account for communication and network delay. We additionally require a verifiability property:

Definition 24 (Verifiability)**.** A protocol $\mathsf{McFly} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Vrfy})$ is verifiable for an NP language $\mathcal{L}_0$ with witness relation $\mathcal{R}_0$, if $(\mathsf{Prove}, \mathsf{Vrfy})$ is a NIZK proof system for a language $\mathcal{L}'$ given by the following induced relation $\mathcal{R}'$:

$$(V = (\mathsf{vk}_1, \dots, \mathsf{vk}_n), d, \mathsf{ct}), (m, r, w_0)) \in \mathcal{R}' \Leftrightarrow$$
$$\mathsf{ct} = \mathsf{McFly.Enc}(1^\lambda, m, d; r, V) \wedge (m, w_0) \in \mathcal{R}_0.$$

Here, $\mathsf{Enc}(\dots; r, V)$ denotes, that the randomness used is r and the keys obtained from the blockchain are V.

Note that this guarantees that (1) a receiver of a verifying pair (ct, π) can be sure to retrieve an output in $\mathcal{L}_0$ after block d was made and (2) outputting π alongside ct reveals no further information.

4.2 Protocol Description

Let SWE be a verifiable SWE, $\mathsf{COM} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Vrfy})$ be a Pedersen commitment, H be the hash function in $\mathcal{BC}$ and H_2 be another hash function. H, H_2 are implicitly made available in all calls to SWE, which is set up for parameters $t = n/2$ out of n. k is the upper bound on the message lengths for SWE.[9] We now describe McFly:

Protocol McFly

$\mathsf{Setup}(1^\lambda)$: Return $\mathsf{COM.Setup}(1^\lambda)$.

$\mathsf{McFly.Enc}^{\mathcal{BC}}(1^\lambda, m, d)$:

- Get the keys V by calling $\mathsf{QueryKeys}$ to $\mathcal{BC}$.
- Split $m = (m_i)_{i \in [\ell]}$ where m_i are from $\{0, \dots, 2^k - 1\}$ with $m = \sum_{i \in [\ell]} 2^{(i-1)k} m_i$.
- $\mathsf{ct} \leftarrow \mathsf{SWE.Enc}(1^\lambda, V, (H(d))_{i \in [\ell]}, (m_i)_{i \in [\ell]})$.
- Output ct.

$\mathsf{McFly.Dec}^{\mathcal{BC}}(\mathsf{ct}, d)$:

- If $\mathsf{QueryTime}$ returns less than d, abort.
- Get (σ, U) by calling $(\mathsf{QueryAt}, d)$ and V by calling $\mathsf{QueryKeys}$ to $\mathcal{BC}$.
- Call $(m_i)_{i \in \ell} \leftarrow \mathsf{SWE.Dec}(\mathsf{ct}, (\sigma)_{i \in [\ell]}, U, V)$.
- Output $m = \sum_{i \in [\ell]} 2^{(i-1)k} m_i$.

$\mathsf{McFly.Prove}^{\mathcal{BC}}(1^\lambda, \mathsf{CRS}, \mathsf{ct}, m, d, w_0, r)$:

- Get the keys V by calling $\mathsf{QueryKeys}$ to $\mathcal{BC}$.
- Split $m = (m_i)_{i \in [\ell]}$ s.t. $m = \sum_{i \in [\ell]} 2^{(i-1)k} m_i$.
- Output $\pi \leftarrow \mathsf{SWE.Prove}(\mathsf{CRS}, V, (H(d))_{i \in [\ell]}, \mathsf{ct}, (m_i)_{i \in [\ell]}, w_0, r)$.

$\mathsf{McFly.Vrfy}^{\mathcal{BC}}(\mathsf{CRS}, \mathsf{ct}, \pi, d)$:

- Get the keys V by calling $\mathsf{QueryKeys}$ to $\mathcal{BC}$.
- Output $b \leftarrow \mathsf{SWE.Vrfy}(\mathsf{CRS}, V, (H(d))_{i \in [\ell]}, \mathsf{ct}, \pi)$

4.3 Proofs

In this section we show that the McFly protocol satisfies the definitions of Section 4.1.

Theorem 2. McFly *is correct, given that* SWE *has robust correctness.*

[9] The size limit is required for efficient decryption, see Section 3.

Proof. Let a parameter λ, a message m^*, a depth d and an algorithm $\mathcal{A}$ be given. Let $\mathsf{ct} \leftarrow \mathsf{Enc}^{\mathcal{BC}}(1^\lambda, m^*, d)$ at any point.

We show that if $\mathsf{McFly}^{\mathcal{BC}}.\mathsf{Dec}(\mathsf{ct}, d)$ is run, when the number of finalized blocks $\mathcal{BC}.\mathsf{QueryTime}$ is greater or equal d, it outputs m. By construction, we have $\mathsf{ct} \leftarrow \mathsf{SWE.Enc}(1^\lambda, V, (H(d))_{i\in[\ell]}, (m_i^*)_{i\in[\ell]})$, where V is the (static) set of keys obtained from $\mathcal{BC}$ and $(m_i^*)_{i\in[\ell]}$ is the result of splitting m^* into chunks from $\{0, \ldots, 2^k - 1\}$.

In our call to Dec, since $\mathcal{BC}.\mathsf{QueryTime}$ is greater or equal d, we can call $(\mathsf{QueryAt}, d)$ and receive (σ, U), which is by definition, such that

$$(\sigma, U) = (\mathsf{Sig}'.\mathsf{Agg}((\sigma_i)_{i\in S}, (\mathsf{vk}_i)_{i\in S}), (\mathsf{vk}_i)_{i\in S})$$

, where for all $i \in [S]$ $\mathsf{Sig}'.\mathsf{Vrfy}(\mathsf{vk}_i, H(d), \sigma_i) = 1$. By correctness of Sig', it holds $\mathsf{Sig}'.\mathsf{AggVrfy}(\sigma, U, (H(d))_{i\in[S]}) = 1$. We then call $m \leftarrow \mathsf{SWE.Dec}(\mathsf{ct}, (\sigma)_{i\in[\ell]}, U, V)$.

Now, if there was an index ind such that $m_{\mathsf{ind}} \neq m_{\mathsf{ind}}{}^*$, forwarding that ind, V, U, $(m_i)_i$, $(T_i)_i$ and σ to the experiment for robust correctness of SWE would constitute a winning adversary. Thus, except with negligible probability $m = m^*$, concluding the proof.

□

Theorem 3. McFly *is secure given that* SWE *is secure and* H *is collision resistant.*

Proof. Let λ, committee size $n = poly(\lambda)$ and a corruption threshold $c < n/2$ be given. Let us assume towards contradiction, that there is an adversary $\mathcal{A}$ with non-negligible advantage ε in $\mathsf{Exp}_{\mathsf{Lock}}(\mathcal{A}, 1^\lambda)$.

First, we discuss a hybrid game $\mathbf{H}_1$: It corresponds to the real experiment, but once we received d from $\mathcal{A}$, if $\mathcal{A}$ has received a signature on $H(d)$ before, we abort. If they query on a signature on that message afterwards, we also abort. If $\mathcal{A}$ had a non-negligible advantage in differentiating H_1 from the experiment, they would have to have a non-negligible advantage in causing an abort. If this were the case, we could directly build a reduction against collision resistance of H.

Thus, we now assume we have an adversary who wins in $\mathbf{H}_1$ with non-negligible probability. We describe a reduction to security of SWE for $t = n/2$ out of n for the set of indices $S = [n]$ at which the challenge message is included. W.l.o.g. we assume n is even.

- The reduction gets access to H_{pr} from the experiment and sets up $\mathcal{BC}$ with access to H_{pr}.
- It computes $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$ and outputs it to $\mathcal{A}$.
- It honestly simulates $\mathcal{BC}$ to $\mathcal{A}$, except in the way it generates the keys and answers signing queries.
- In the initialization:
  - Let C' be the indices of malicious keys chosen by $\mathcal{A}$ and set $\bar{C} = [n] \setminus C'$. Let $V' = (\mathsf{vk}_i)_{i\in C'}$ be the malicious keys and π_i the corresponding proofs.
  - The reduction receives $n/2 + 1$ keys $V_E = \mathsf{vk}_i'$ from the experiment and sets the first $n/2 + 1$ of the honest keys $(\mathsf{vk}_i)_{i\in\bar{C}}$, to be these vk_i'. If the adversary chose $|C'| < n/2 - 1$, the remaining honest keys are generated by $\mathsf{Sig}'.\mathsf{KeyGen}$ and saved as V_R. The proofs of validity for keys in $\mathcal{V}_E$ are received from the experiment and for $\mathcal{V}_R$, the reduction computes them honestly.
- For signing:
  - For keys in V_R we sign honestly.
  - For keys in V_E we relay signing queries to the experiment.
- Then, we output $V_R \cup V'$ as challenge keys to the experiment, where we receive the validity proofs for V' from $\mathcal{A}$. These verify by definition of $\mathcal{BC}$.

- We receive messages m_0, m_1 and a depth $d > 0$ from $\mathcal{A}$. We choose $(m_i^0)_i \in [\ell]$ as a split of m_0 and $(m_i^1)_i \in [\ell]$ as a split of m_1. We output $(m_i)_i$ and $T_i = d$ for all $i \in [\ell]$ to the experiment.
- We receive back a ciphertext ct that we forward to $\mathcal{A}$.
- Now, if $\mathcal{A}$ outputs a bit b in time, we output it to the experiment. Otherwise, we output a random bit.

Note, that by our abort conditions, we have not asked for a signature on T_{ind} before outputting it nor afterwards, causing the interaction with the experiment to run through.

If the experiment chooses bit b', our output is identically distributed to running $\mathbf{H}_1$ with $b = b'$. Since we also guess randomly, if $\mathcal{A}$ does not output a bit, that means, that we get the same advantage as $\mathcal{A}$ does in $\mathbf{H}_1$. Assuming security of SWE, that concludes the proof.

□

Theorem 4. McFly *is verifiable, given that* SWE *is a verifiable SWE.*

This follows immediately by the definition of McFly.Enc and verifiability of SWE, as we only invoke the underlying NIZK proof system.

4.4 Integration with Casper

As a concrete example, we discuss the modifications necessary to the Ethereum 2.0 Altair [30] protocol running with the Casper finality layer [15]. In the Casper protocol [15] the committees are randomly sampled and become responsible for finalizing blocks for some period of time, called an epoch. The members of the committee can cast votes on the blocks they believe to be final by signing the blocks using their signing keys. Once a majority of the committee votes on the same block it becomes final. As new committees are chosen only one epoch in advance and each epoch lasts for about 6.4 minutes, we can at most encrypt to 12.8 minutes into the future. Moreover, a final block is only produced roughly after every epoch duration. So while we have a near-constant block production rate, the horizon choice is essentially limited to the start of the next epoch. However, an extension of the above is possible in the current version of Ethereum Altair [30, 14]. There, the so-called sync committees have a lifetime of roughly 27 hours and are appointed "one round" in advance.[10] These committees periodically sign the header of the newest block to enable faster verification for light clients. We believe that a horizon of 27 hours is sufficient for most applications, including the ones we will describe in Section 4.5. We stress however that the security implications for the concept of sync committees were not formally analyzed in Ethereum Altair [30, 14].

We describe the modifications needed on Ethereum 2.0 in order to instantiate McFly on top of it:

- Since Casper already uses BLS signatures, there are two possible alternatives. (1) The committee of Casper adopts the aggregation mechanism described in Section 5, or (2) decryptors obtain the unaggregated signatures themselves (which is already possible with the Ethereum beacon chain).[11] In the latter case, signature aggregation can be performed locally at the decryptor and hence no modifications are necessary. One could also envision dedicated aggregation servers when the system is widely adopted.
- For each finalized block the (sync) committee additionally signs a counter r that represents the number of finalized blocks (or slot number).

[10] When a new sync committee becomes active, the next one is sampled.

[11] See https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/p2p-interface.md#attestation-subnets

- The public keys of the committee members must have a proof of knowledge. This can be achieved, e.g., by registering the keys with a PKI.

Extension for Dynamic Committees In our model, we assumed static committees. However, a fixed committee can be a security risk for a finality layer deployed in the wild as committee members usually become target of attacks. ence, finality layers advocate for a short-lived dynamic committee [23], mitigating this risk.

We can safely regard a committee as known and static during its lifetime. Thus, our model naturally extends as long as we only encrypt messages as far into the future as the committees are currently known.

Moreover, in the period where the current committee is about to be replaced and the next committee is not yet known, the limitation becomes more severe. However, this can be mitigated by considering interleaved committees for the finality layer; one committee starts to be active at the middle of the lifetime of the other, and both committees independently run the protocol - this ensures a smooth transition between committees.

4.5 Applications

In this section we describe two applications that can be easily built using the McFly protocol.

Decentralized Auctions An auction is a process of trading goods for bids, and it is run among an arbitrary number of bidders, and an auctioneer. First, in a bidding phase, all the bidders submit their bids. Then, in a final phase the auctioneer, after checking all bids announces the winner. If the auctioneer is not fully honest, it is easy to see that the outcome of the auction cannot be trusted. Therefore, protocols for decentralized auctions (or auctioneer-free protocols) are of great interest. Here we focus on sealed-bid auctions that are run exclusively on a blockchain, i.e., the communication of the parties happens only through blockchain transactions. Additionally, the set of parties running the blockchain and bidding on the auction can overlap, making it profitable for malicious parties to, e.g., not include bids that are greater than their own.

One difficulty in realizing decentralized auctions in blockchains, as noted in [21], is that a bid transaction should be self-contained, i.e., the auction protocol should be able to terminate and determine the correct winner after collecting all bid transactions. Particularly, this rules out solutions where the bid transaction is a commitment to the real bid, and later an opening to this commitment is sent. The reason is that if no opening is ever received, it is not possible to determine if the opening was suppressed by malicious parties in the blockchain or a malicious bidder is refusing to open his bid to DoS the auction. This is handled in [21] by including in the bidding transaction a time-lock puzzle containing the opening of the commitment to the bid; whenever an opening is not received, the auction can still terminate by solving the time-lock puzzles and retrieving the bids. This comes with all the disavantages of timelock puzzles discussed in the introduction.

In contrast, with the proposed time-release encryption mechanism, we can easily realize the same decentralized auction as [21] without any overhead on the blockchain, by simply encrypting the bids for a future block. We give next a high level description of this. Assume we want the bidding phase to start at final block number r in the underlying blockchain, and span through a sequence of ℓ final blocks. During the bidding phase, all parties will encrypt their bids with respect to the committee members' verification keys and to the counter $r + \ell$. After this final block is signed by the committee, the signature can be used to decrypt all the bids, making it possible to publicly verify who the auction winner is. Note that with this simple approach there is no bid privacy for the losing bids, as all the bids are opened at the end of the auction run. Moreover, if the underlying blockchain is incentive-compatible, the results of [21] show that the encrypted bids will always be included in the blockchain given that it pays the required fees.

Randomness Beacons A randomness beacon is a tool that provides public randomness at predefined intervals. Blockchains can be a great source of randomness to build decentralized randomness beacons, where distrustful parties contribute to the randomness output by the beacon. One known problem with using blockchains for building randomness beacons is that malicious parties could somehow manipulate the contents to be included in the blockchain as a way to introduce bias in the beacon output. For that, some solutions [32, 7, 2] leverage the use of verifiable-delay functions (VDF) to delay the output of the beacon, such that malicious parties do not have enough time at their disposal to be able to bias the randomness. However, VDFs carry the same drawbacks as time-lock puzzles; they are wasteful and it is usually hard to set parameters that result in a secure and usable system. As the role of VDFs in these randomness beacon constructions is simply to hide information from the parties for a predetermined period of time, with only minor modifications one could replace the VDFs for our time-release encryption mechanism in [32, 7, 2] to get randomness beacons with almost no overhead on the blockchain.

5 BLS signatures with modified aggregation

In this section we describe a modified BLS signature scheme. Looking ahead, this scheme will be used as a building block when we instantiate SWE in Section 6. The key generation, signing and verification algorithms will be identical to regular BLS signatures.

5.1 Construction

Our system parameters include two base groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order p with generators g_1, g_2 which have a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ into a target group $\mathbb{G}_T$ with generator g. Also we assume full-domain hash functions $H : \{0,1\}^* \to \mathbb{G}_1$ and $H_2, H_{pr} : \{0,1\}^* \to \mathbb{Z}_p$. Let (Schnorr.$\mathsf{Prove}$,Schnorr.$\mathsf{Valid}$) be the non-interactive variant of the well-known Schnorr proofs due to Fischlin [26] discussed in Section 2.

The algorithms of our aggregatable multi-signature scheme $\mathsf{Sig}' = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Agg}, \mathsf{AggVrfy}, \mathsf{Prove}, \mathsf{Valid})$ are given as follows.

$\mathsf{Sig}'.\mathsf{KeyGen}(1^\lambda)$:

- Randomly pick $x \leftarrow_\$ \mathbb{Z}_p$.
- Output $(\mathsf{vk} = {g_2}^x, \mathsf{sk} = x)$.

$\mathsf{Sig}'.\mathsf{Sign}(\mathsf{sk}, T)$:

- Output $H(T)^{\mathsf{sk}}$.

$\mathsf{Sig}'.\mathsf{Vrfy}(\mathsf{vk}, T, \sigma)$:

- Compute $h = H(T)$.
- If $(e(\sigma, g_2) = e(h, \mathsf{vk}))$, output 1, else output 0.

$\mathsf{Sig}'.\mathsf{Agg}((\sigma_1, \ldots, \sigma_k), (\mathsf{vk}_1, \ldots, \mathsf{vk}_k))$:

- Compute $\xi_i = H_2(\mathsf{vk}_i)$ for $i \in [k]$.
- Compute $L_i = \prod_{j \in [k], i \neq j} \frac{-\xi_j}{\xi_i - \xi_j}$ for $i \in [k]$.
- Output $\sigma \leftarrow \prod_{i \in [k]} \sigma_i^{L_i}$.

$\mathsf{Sig}'.\mathsf{AggVrfy}(\sigma, (\mathsf{vk}_1, \ldots, \mathsf{vk}_k), (T_1, \ldots, T_k))$:

- If $e(\sigma, g_2) = \prod_{i\in[k]} e(H(T_i), \mathsf{vk}_i)^{L_i}$, output 1. Output 0 otherwise.

$\mathsf{Sig}'.\mathsf{Prove}(\mathsf{vk}, \mathsf{sk})$:

- Output Schnorr.$\mathsf{Prove}^{H_{pr}}(\mathsf{vk}, \mathsf{sk})$.

$\mathsf{Sig}'.\mathsf{Valid}(\mathsf{vk}, \pi)$:

- Output Schnorr.$\mathsf{Valid}^{H_{pr}}(\mathsf{vk}, \pi)$.

It is shown in the random oracle model in [26], that (Schnorr.Prove,Schnorr.Valid) fulfills the requirements of an online-extractable zero-knowledge proof of knowledge for the relation $\mathcal{K} = \{(g^x, x) : x \in \mathbb{Z}_p\}$. We use Fischlin, rather than the Fiat-Shamir-transformation [25], because the latter doesn't guarantee extraction of the witness in polynomial time.

Alternatively, a standard proof of possession as used in [36] can be used in the construction of Sig'. This still allows us to use Sig' in the further constructions of SWE and McFly. The proof of possession achieves weaker but similar guarantees to the proof of knowledge requirement that are sufficient for the further proofs, if we additionally make the knowledge of exponent assumption. As the construction and proofs are more straight-forward using Schnorr, we will focus on this case in the main body. We provide a discussion on using proofs-of-possession in Appendix A. The resulting algorithms using proofs-of-possession would be:

$\mathsf{Prove}(\mathsf{vk}, \mathsf{sk})$:

- Outputs $\mathsf{Sig}'.\mathsf{Sign}^{H_{pr}}(\mathsf{sk}, \langle\mathsf{vk}\rangle)$.

$\mathsf{Valid}(\mathsf{vk}, \pi)$:

- Check whether $\mathsf{Sig}'.\mathsf{Vrfy}^{H_{pr}}(\mathsf{vk}, \langle\mathsf{vk}\rangle, \pi)$.
- If so, output 1, else $\perp$.

5.2 Proofs

We will now show that Sig' fulfills the requirements for aggregatable multi-signatures stated in Section 2. We have discussed that $(\mathsf{Sig}'.\mathsf{Prove}, \mathsf{Sig}'.\mathsf{Valid})$ constitutes a valid online-extractable proof of knowledge for the key relation already.

Theorem 5. Sig' *is correct.*

Proof. The first part of correctness follows directly from [11] as the algorithms and definitions are identical to standard BLS.

Let $\lambda \in \mathbb{N}$, $k = poly(\lambda)$, messages $T_1, \ldots, T_k$ a set of public keys $V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_k)$ and signatures $\sigma_1, \ldots, \sigma_k$ be given such that $\mathsf{Vrfy}(\mathsf{vk}_i, T_i, \sigma_i) = 1$ for $i \in [k]$. This means it holds $e(\sigma_i, g_2) = e(H(T_i), \mathsf{vk}_i)$ for $i \in [k]$. Let $\sigma \leftarrow \mathsf{Agg}((\sigma_1, \ldots, \sigma_k), V)$. We need to show $\mathsf{AggVrfy}(\sigma, V, (T_1, \ldots, T_k)) = 1$.

By construction, the aggregated multi-signature is $\sigma = \prod_{i\in[k]} \sigma_i^{L_i}$ for L_i as defined in our algorithm. Now, in our call to AggVrfy, it holds

$$e(\sigma, g_2) = e(\prod_{i\in[k]} \sigma_i^{L_i}, g_2) = \prod_{i\in[k]} e(\sigma_i, g_2)^{L_i} = \prod_{i\in[k]} e(H(T_i), \mathsf{vk}_i)^{L_i}.$$

Therefore, the output is 1. □

Theorem 6. *Assume that H is modelled as a random oracle.* Sig' *is unforgeable, given that the computational Co-Diffie-Hellman assumption holds for* $(\mathbb{G}_1, \mathbb{G}_2)$.

Proof. Our proof takes some ideas from [36] and [10], but mainly works due to the extractability of Schnorr. We will regard adversaries $\mathcal{A}$ that will be allowed to make only polynomially many q_S queries for signatures.

We assume that $\mathcal{A}$ has non-negligible winning probability ε and will only output a forge $(T_1, \ldots, T_k), (\mathsf{vk}_2, \ldots, \mathsf{vk}_k), (\pi_2, \ldots, \pi_k), \sigma^*$ where $\mathsf{Valid}(\mathsf{vk}_i, \pi_i) = 1$ for all $i \in \{2, \ldots, k\}$ and T_1 was not queried to the signing oracle, otherwise they would not be able to win in the unforgeability experiment. Assuming control over the random oracle, we can use the fact that $(\mathsf{Prove}, \mathsf{Valid})$ constitutes an online-extractable proof of knowledge.

We define a reduction against co-CDH:

The challenger receives a tuple $g_1, h_1 = g_1^x, g_2, h_2 = g_2^x, h$ with the public parameters g_1, g_2 as generators and is required to output h^x.

The experiment executes $(H_{pr}, \tau_0) \leftarrow S(0, 1^\lambda)$ and provides oracle access to H_{pr} to $\mathcal{A}$.

The adversary gets access to the publicly known $\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, H_2$. Queries to the hash function H are being programmed by the reduction.

The reduction provides as public key $\mathsf{vk}^* = h_2$ to $\mathcal{A}$.

Any queries to H are answered as follows:

- If the message T was previously queried, we respond as before.
- Else, with probability δ we select its hash as $h \cdot g_1^\theta$ for random $\theta \leftarrow_\$ [p]$ and save T to a special list C.
- Otherwise, we select its hash as g_1^θ for random $\theta \leftarrow_\$ [p]$
- In both cases, we save $A[T] = \theta$

If $\mathcal{A}$ queries for a proof of knowledge for vk^* we call $(H_1, \pi^*, \tau') \leftarrow S(1, \mathsf{vk}^*, \tau, \mathtt{YES})$, respond with π^* and replace the oracle H_{pr} by H_1 in future calls. By zero-knowledge of Schnorr, this is indistinguishable from the real experiment's output except with negligible probability. We note, that all $\mathsf{vk} \in \mathbb{G}_2$ actually have a valid secret key, making the $\mathtt{YES}$ call justified.

Any queries to the signing oracle for a message T under vk^* are answered as follows:

- We determine $H(T)$.
- If T is in the special list C, we abort.
- Otherwise, we know the hash $H(T) = g_1^\theta$.
- We output as signature h_1^θ. Since $h_1 = g_1^x$ for some x such that $\mathsf{vk} = h_2 = g_2^x$, this is simply $g_1^{\theta x} = H(T)^x$, which is a valid signature under vk.

Once we receive $(T_1, \ldots, T_k), (\mathsf{vk}_2, \ldots, \mathsf{vk}_k), (\pi_2, \ldots, \pi_k), \sigma^*$ from $\mathcal{A}$,

- For $i \in \{2, \ldots, k\}$, we call the PPT extractor $\mathcal{E}(\mathsf{vk}_i, \pi_i, Q_\mathcal{A})$ where $Q_\mathcal{A}$ are the queries to H_{pr} so far.
- Since $\mathsf{Valid}(\mathsf{vk}_i, \pi_i)$ holds by assumption, we can extract the sk_i except with negligible probability and save them to a table $P[\mathsf{vk}_i] = \mathsf{sk}_i$. If we fail to extract for any index, we abort.
- We check whether T_1 is on the list C and whether σ^* is a valid forge. If this is not the case, we abort.
- If any of the keys $\mathsf{vk}_2, \ldots, \mathsf{vk}_n$ is equal to vk^*, the adversary may not have asked for our simulated proof on vk^* and therefore must have given a proof of their own which we extracted from - we have $x = P[\mathsf{vk}_i]$ for that key by definition and thus can output h^x directly.
- Otherwise, it holds $e(\sigma^*, g_2) = \prod_{i \in [k]} e(H(T_i), \mathsf{vk}_i)^{L_i}$ where we consider $\mathsf{vk}^* = \mathsf{vk}_1$.

- Now for $i \in \{2, \ldots, k\}$, we can make partial signatures $\sigma_i = H(T_i)^{P[\mathsf{vk}_i] L_i}$ with $e(\sigma_i, g_2) = e(H(T_i), \mathsf{vk}_i)^{L_i}$.
- We set σ' to be $\sigma^* / \prod_{i \in \{2,\ldots,k\}} (\sigma_i)$. Now, it clearly holds $e(\sigma', g_2) = e(H(T_1), \mathsf{vk}_1)^{L_1}$. Therefore, $\sigma' = (h \cdot g_1^{A[T_1]})^{x L_1}$ and we output $\left(\frac{\sigma'}{h_1^{A[T_1]L_1}}\right)^{-L_1} = h^x$. This holds as $h_1 = g_1^x$.

Clearly, if no abort occurs, the output is indeed h^x.

Now, what is the success probability? Assume the adversary $\mathcal{A}$ has advantage ε in winning the unforgeability experiment. If they win, they can either query us for a proof on vk^* or be able to include vk^* in their forge.

$\mathcal{A}$ can only succeed, if its combined probability of successfully registering all keys $\mathsf{vk}_2, \ldots, \mathsf{vk}_k$ it chooses is at least ε, making every one of these probabilities non-negligible. By extractability of our proof of knowledge and a union bound, this means we can extract all secret keys in polynomial time except with negligible probability. We note that since the hashes and vk^* are distributed uniformly random, this looks indistinguishable from the real experiment for $\mathcal{A}$ unless they request a signature for one of the messages where T is in C. This probability can be bounded by $(1-\delta)^{q_S}$, assuming $\mathcal{A}$ only queries for messages once, as the probability is clearly independent for every message requested. Conditioned on no such request being made, we have a probability of ε of the adversary winning. Since the hash(es) which we created as $h \cdot g_1^{\theta}$ are i.i.d. in the view of $\mathcal{A}$, we then have a probability of δ of the first message m_1 in fact being such that we don't abort.

This gives us a winning probability negligibly worse than $(1-\delta)^{q_S} \cdot \delta \cdot \varepsilon$. By appropriately choosing $\delta = 1/q_S$ we get $(1 - 1/q_S)^{q_S} \cdot 1/q_S \cdot \varepsilon \geq 0.1/q_S \cdot \varepsilon$, assuming that $q_S \geq 2$, as $(1-1/x)^x$ converges to $1/e$ in a strictly increasing manner. Therefore the advantage of the reduction will be non-negligible, if ε is not negligible.

The reduction is clearly running in polynomial time if $\mathcal{A}$ is.

□

6 Construction of Signature-based Witness Encryption

In this section we provide an instantiation for a t-out-of-n SWE scheme that is compatible with the modified BLS signature scheme defined in the previous section. Its security is based on the bilinear Diffie-Hellman assumption. Our construction is specifically optimized to push as many operations as possible into the source group $\mathbb{G}_2$. This leads to significant performance improvements over a naive approach if we choose $\mathbb{G}_2$ to be the one of the two source groups for which group operations are cheaper.

6.1 Construction

We mention two constructions: The base construction, which appeared in an earlier version of this manuscript as the only construction, and has the limitation, that the inputs T_i must all be distinct, but has slightly smaller cipher texts. We add an enhanced construction in this version of the paper, which overcomes this obstacle by adding randomness and should usually be preferred in practice.

Protocol SWE' - Enhanced construction

$\mathsf{SWE}'.\mathsf{Enc}(1^\lambda, (\mathsf{vk}_j)_{j\in[n]}, (T_i)_{i\in[\ell]}, (m_i)_{i\in[\ell]})$:

- Choose random $r, r_j \leftarrow_\$ \mathbb{Z}_p$ for $j \in \{0, \dots, t-1\}$.
- Let $f(x) = \sum_{j=0}^{t-1} r_j \cdot x^j$. This will satisfy $f(0) = r_0$.
- For $j \in [n]$, set $\xi_j = H_2(\mathsf{vk}_j)$, $s_j = f(\xi_j)$.
- For $i \in [\ell]$ choose random $\alpha_i \leftarrow_\$ \mathbb{Z}_p$.
- Compute $c = g_2^r$, $a_i = c^{\alpha_i}$, $t_i = H(T_i)^{\alpha_i}$ for $i \in [\ell]$.
- Choose $h \leftarrow \mathbb{G}_2$ uniformly at random.
- Compute $c_0 = h^r \cdot g_2^{r_0}$.
- For $j \in [n]$, compute $c_j = \mathsf{vk}_j^r \cdot g_2^{s_j}$.
- For $i \in [\ell]$, set $c_i' = e(t_i, g_2^{r_0}) \cdot g_T^{m_i}$.
- Output $\mathsf{ct} = (h, c, c_0, (c_j)_{j\in[n]}, (c_i', a_i, t_i)_{i\in[\ell]})$.

$\mathsf{SWE}.\mathsf{Dec}(\mathsf{ct}, (\sigma_i)_{i\in[\ell]}, U, V)$:

- Parse $\mathsf{ct} = (h, c, c_0, (c_j)_{j\in[n]}, (c_i', a_i, t_i)_{i\in[\ell]})$.
- Parse $V = (\mathsf{vk}_1, \dots, \mathsf{vk}_n)$, $U = (\mathsf{vk}_1', \dots, \mathsf{vk}_k')$.
- If $k < t$ or $U \nsubseteq V$, abort.
- Define as I the indices $j \in [n]$ s.t. $\mathsf{vk}_j \in U$.
- Compute $\xi_j = H_2(\mathsf{vk}_j)$ for $j \in I$.
- Compute $L_j = \prod_{i\in I, i\neq j} \frac{-\xi_i}{\xi_j - \xi_i}$ for $j \in I$.
- Compute $c^* = \prod_{j\in I} c_j^{L_j}$.
- For $i \in [\ell]$, compute
  $z_i = c_i' \cdot e(\sigma_i, a_i)/e(t_i, c^*)$.
- For $i \in [\ell]$, compute $m_i' = \mathsf{dlog}_{g_T}(z_i)$.
- Output $(m_i')_i$.

Notice here, that we only do the expensive computation of c^* once. This is on the condition that in the use of our protocol, the sets of signers are the same for all T_i. If they aren't or only some of the multi-signatures σ_i are given, it is still possible to compute the m_i for which we have signatures on T_i, but we may have to compute c^* for all relevant sets U of signers.
In order to enable an efficient interface to a commitments scheme (see Section 7.2), the redundant terms h and c_0 are computed by encryption. Based on this, we will subsequently add algorithms $\mathsf{SWE}.\mathsf{Prove}, \mathsf{SWE}.\mathsf{Vrfy}$ that make SWE verifiable, but focus on the core functionality for now. Further, we choose m_i from $\mathbb{Z}_p$ to enable the usage of efficient bulletproofs to go with these commitments. Note that the extraction of the discrete logarithm does not cause a large overhead as we use the baby-step giant-step methodology (compare Section 8).

6.1.1 Base Construction

In the base construction, we did not rerandomize the c_i' with fresh α_i for each slot $i \in [\ell]$, this can be seen as SWE' with setting all $\alpha_i = 1$. This is the originally proposed scheme:

Protocol SWE - base construction

$\mathsf{SWE}.\mathsf{Enc}(1^\lambda, (\mathsf{vk}_j)_{j\in[n]}, (T_i)_{i\in[\ell]}, (m_i)_{i\in[\ell]})$:

- Choose random $r, r_j \leftarrow_\$ \mathbb{Z}_p$ for $j \in \{0, \dots, t-1\}$.

- Let $f(x) = \sum_{j=0}^{t-1} r_j \cdot x^j$. This will satisfy $f(0) = r_0$.
- For $j \in [n]$, set $s_j = f(\xi_j)$, where $\xi_j = H_2(\mathsf{vk}_j)$.
- Compute $c = g_2^r$.
- Choose $h \leftarrow \mathbb{G}_2$ uniformly at random.
- Compute $c_0 = h^r \cdot g_2^{r_0}$.
- For $j \in [n]$, compute $c_j = \mathsf{vk}_j^r \cdot g_2^{s_j}$.
- For $i \in [\ell]$, set $c'_i = e(H(T_i), g_2^{r_0}) \cdot g_T^{m_i}$.
- Output $\mathsf{ct} = (h, c, c_0, (c_j)_{j\in[n]}, (c'_i)_{i\in[\ell]})$.

$\mathsf{SWE.Dec}(\mathsf{ct}, (\sigma_i)_{i\in[\ell]}, U, V)$:

- Parse $\mathsf{ct} = (h, c, c_0, (c_j)_{j\in[n]}, (c'_i)_{i\in[\ell]})$.
- Parse $V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$.
- Parse $U = (\mathsf{vk}'_1, \ldots, \mathsf{vk}'_k)$. If $k < t$ or $U \not\subseteq V$ abort.
- Define as I the indices $j \in [n]$ such that $\mathsf{vk}_j \in U$.
- Compute $\xi_j = H_2(\mathsf{vk}_j)$ for $j \in I$.
- Compute $L_j = \prod_{i\in I, i\neq j} \frac{-\xi_i}{\xi_j - \xi_i}$ for $j \in I$.
- Compute $c^* = \prod_{j\in I} c_j^{L_j}$.
- For $i \in [\ell]$, compute
  $z_i = c'_i \cdot e(\sigma_i, c)/e(H(T_i), c^*)$.
- For $i \in [\ell]$, compute $m'_i = \mathsf{dlog}_{g_T}(z_i)$.
- Output $(m'_i)_i$.

6.2 Efficiency

We will briefly analyze the number of group operations in each group required for encryption and decryption. We regard the number of n and ℓ to be fixed and give upper bounds on the operations needed. We require no multiplications or exponentiations in $\mathbb{G}_1$. In practice $\mathbb{G}_1$ and $\mathbb{G}_2$ can be reversed if $\mathbb{G}_1$ is more efficient in a given implementation.

	SWE encryption	SWE decryption	SWE′ encryption	SWE′ decryption
evaluations of H, H_2	ℓ, n	ℓ, n	ℓ, n	$0, n$
multiplications, exponentiations in $\mathbb{G}_2$	$n+1, 2n+3$	$n-1, n$	$n+1, 2\ell+2n+3$	$n-1, n$
multiplications, exponentiations in $\mathbb{G}_T$	ℓ, ℓ	$2\ell, 0$	ℓ, ℓ	$2\ell, 0$
pairing evaluations	ℓ	2ℓ	ℓ	2ℓ
dlog in $\mathbb{G}_T$	0	ℓ	0	ℓ

The cost of allowing repeated reference messages T_i in SWE′ is 2ℓ additional exponentiations in the cheaper source group $\mathbb{G}_2$ and 2ℓ more ciphertext parts in the output. This is a relatively mild overhead. A reference implementation was made for the base scheme SWE and evaluations can be found in Section 8.

6.3 Proofs

We will now show SWE′ fulfills the requirements for a signature-based witness encryption and SWE does so, if the reference messages T_i involved are all distinct.

Theorem 7. SWE *and* SWE' *for the signature scheme* Sig' *both have robust correctness, given that* H_2 *is collision resistant.*

Proof. Let $\lambda \in \mathbb{N}$, $\ell = poly(\lambda)$ be given. Let us assume towards contradiction, that there is an adversary $\mathcal{A}$ with non-negligible winning probability against the experiment.

Let us consider a hybrid $\mathbf{H}_1$: It is identical to the experiment, except if $H_2(\mathsf{vk}_i) = H_2(\mathsf{vk}_j)$ for any $i, j \in [n], i \neq j$, we abort. Clearly, except with negligible probability running $\mathbf{H}_1$ with $\mathcal{A}$ has the same outcome as the original experiment. Otherwise, we could build a reduction against the collision resistance of H_2.

We now show, that in $\mathbf{H}_1$, the probability of winning for the adversary is 0. Let any index $\mathsf{ind} \in [\ell]$, keys $V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$, a subset $U \subseteq V$ with $|U| \geq t$, reference messages $(T_i)_{i\in[\ell]}$, messages $(m_i)_{i\in[\ell]}$ and $(\sigma_i)_{i\in[\ell]}$ be given by $\mathcal{A}$. Let I be the set of all indices i for which $\mathsf{vk}_i \in U$.

We note, that since g_2 is a generator of $\mathbb{G}_2$, there exist x_i such that $\mathsf{vk}_i = g_2^{x_i}$ for $i \in [n]$. We assume $\mathsf{AggVrfy}(\sigma_{\mathsf{ind}}, U, (T_{\mathsf{ind}})_{i\in[|U|]}) = 1$, that is $e(\sigma_{\mathsf{ind}}, g_2) = \prod_{i\in I} e(H(T_{\mathsf{ind}}), \mathsf{vk}_i)^{L_i}$. Otherwise, $\mathcal{A}$ could not win. We now show that

$$\mathsf{Dec}(\mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, (m_i)_{i\in[\ell]}), (\sigma_i)_{i\in[\ell]}, U, V)_{\mathsf{ind}} = m_{\mathsf{ind}}$$

- For the base construction SWE, the ciphertext $\mathsf{ct} = (h, c, c_0, (c_j)_{j\in[n]}, (c'_i)_{i\in[\ell]}) = \mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, (m_i)_{i\in[\ell]})$ has the relevant components for decryption $c = g_2^r$, $c_j = \mathsf{vk}_j^r \cdot g_2^{s_j}$ for $j \in [n]$ and $c'_i = e(H(T_i), g_2)^{r_0} \cdot g^{m_i}$ for $i \in [\ell]$, where $s_j = f(\xi_j)$ for a polynomial such that $f(0) = r_0$.
- For the enhanced construction SWE', additionally, $a_i = c^{\alpha_i}, t_i = H(T_i)^{\alpha_i}$ are given and we set $c'_i = e(H(T_i), g_2)^{r_0\alpha_i} \cdot g^{m_i}$

Now, in both cases the ξ_j, L_j computed by Dec are identical to those used in Enc and in $\mathsf{Sig}'.\mathsf{Sign}$. Note that since no two distinct $\mathsf{vk}_j \neq \mathsf{vk}_{j'}$ collide under H_2, the support points $\xi_j = H_2(\mathsf{vk}_j)$ are all distinct and $|I| \geq t$ so Lagrange interpolation will correctly recover r_0 from the s_j by computing $r_0 = f(0) = \sum_{j\in I} s_j L_j$. Thus it holds that

$$c^* = \prod_{j\in I} c_j^{L_j} = \left(\prod_{j\in I} \mathsf{vk}_j^{L_j}\right)^r \cdot \prod_{j\in I} g_2^{s_j L_j} = (\mathsf{vk}^*)^r \cdot g_2^{r_0}.$$

where $\mathsf{vk}^* := \prod_{j\in I} \mathsf{vk}_j^{L_j} = g_2^{\sum_{j\in I} x_j L_j}$.

- Now, in the case of the base SWE, it holds for index ind:

$$\begin{aligned} e(H(T_{\mathsf{ind}}), c^*) &= e(H(T_{\mathsf{ind}}), g_2^{r\cdot\sum_{j\in I} x_j L_j} \cdot g_2^{r_0}) \\ &= \prod_{j\in I} e(H(T_{\mathsf{ind}}), \mathsf{vk}_j)^{r\cdot L_j} \cdot e(H(T_{\mathsf{ind}}), g_2)^{r_0} \\ &= e(\sigma_{\mathsf{ind}}, g_2)^r \cdot e(H(T_{\mathsf{ind}}), g_2)^{r_0} \\ &= e(\sigma_{\mathsf{ind}}, c) \cdot e(H(T_{\mathsf{ind}}), g_2)^{r_0}. \end{aligned}$$

Since $c'_{\mathsf{ind}} = e(H(T_{\mathsf{ind}}), g_2)^{r_0} \cdot g_T^{m_{\mathsf{ind}}}$, it follows that $z_{\mathsf{ind}} = c'_{\mathsf{ind}} \cdot e(\sigma_{\mathsf{ind}}, c)/e(H(T_{\mathsf{ind}}), c^*) = g_T^{m_{\mathsf{ind}}}$. It follows for the ind-th output: $m'_{\mathsf{ind}} = \mathsf{dlog}_{g_T}(z_{\mathsf{ind}}) = m_{\mathsf{ind}}$, and robust correctness follows.

- In the case of the enhanced SWE', analogously it holds for index ind:

$$\begin{aligned} e(t_{\mathsf{ind}}, c^*) &= e(H(T_{\mathsf{ind}}), c^*)^{\alpha_{\mathsf{ind}}} = e(\sigma_{\mathsf{ind}}, c)^{\alpha_{\mathsf{ind}}} \cdot e(H(T_{\mathsf{ind}}), g_2)^{r_0 \alpha_{\mathsf{ind}}} \\ &= e(\sigma_{\mathsf{ind}}, a_{\mathsf{ind}}) \cdot e(t_{\mathsf{ind}}, g_2)^{r_0} \end{aligned}$$

  and thus decryption via $z_{\mathsf{ind}} = c'_{\mathsf{ind}} \cdot e(\sigma_{\mathsf{ind}}, a_{\mathsf{ind}})/e(t_{\mathsf{ind}}, c^*) = g_T^{m_{\mathsf{ind}}}$ also yields robust correctness.

□

Theorem 8. *Assume that the hash functions H, H_2, H_{pr} are modelled as random oracles and all references T_i are distinct. Then* SWE *for the signature scheme* Sig' *is secure under the BDH assumption in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$. The security reduction is tight.*

Proof. We actually show a little bit more, namely, that the ciphertexts are essentially pseudorandom at the indices where the challenge messages are included.

Lemma 9. *Specifically, assuming that the hash functions H, H_2, H_{pr} are modelled as random oracles and all references T_i are distinct and under the BDH assumption in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, the experiment $\mathsf{Exp}_{\mathsf{Sec}}(\mathcal{A}, 1^\lambda)$ is indistinguishable from a modified experiment $\mathsf{Exp}'_{\mathsf{Sec}}$, in which, instead of a well-formed $\mathsf{ct} = (h, c, c_0, (c_j)_{j\in[n]}, (c'_i)_{i\in[\ell]}) = \mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, (m_i)_{i\in[\ell]})$, we replace (c'_i) by a random element u_i from $\mathbb{G}_T$ for all challenge indices $i \in SC$.*

As the c'_i are the only parts of ct that depend on m_i^b, this implies security.

Assume that $\mathcal{A}$ is a PPT adversary distinguishing $\mathsf{Exp}'_{\mathsf{Sec}}$ from the real $\mathsf{Exp}_{\mathsf{Sec}}$ with advantage ε. We will construct a PPT distinguisher $\mathcal{D}$ with advantage ε against the BDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$. The complexity of $\mathcal{D}$ is essentially the same as that of $\mathcal{A}$.

Let $I = \{1, \ldots, t-1\}$ be the set of indices of verification keys vk_i which are chosen by the adversary. For each $i \in I$ let $\mathsf{vk}_i = g_2^{x_i}$ where $\mathsf{sk}_i = x_i$. In the following we will assume that the distinguisher $\mathcal{D}$ has access to all the signing keys x_i for $i \in I$. This can be achieved, by using the fact that Sig' has an online-extractable proof of knowledge.

Let SC be the set of indices at which the challenge messages are included in encryption.

The distinguisher $\mathcal{D}$ receives as input a tuple $(g_1, h_1, v, g_2, h_2, w, z)$, where g_1 is a generator of $\mathbb{G}_1$, g_2 is a generator of $\mathbb{G}_2$, $h_1 = g_1^x$ for some $x \in \mathbb{Z}_p$, $h_2 = g_2^x$ for the same $x \in \mathbb{Z}_p$, $v = g_1^\alpha$ for an $\alpha \in \mathbb{Z}_p$ and $w = g_2^r$ for an $r \in \mathbb{Z}_p$. The term $z \in \mathbb{G}_T$ is either of the form $g_t^{\alpha x r}$, in which case we say that this is a BDH tuple, or z is chosen uniformly random from $\mathbb{G}_T$, in which case we say this is a random tuple.

The distinguisher $\mathcal{D}$ simulates the security experiment for SWE, except in the way the verification keys vk_i for $i \in \bar{I} = \{t, \ldots, n\}$ are chosen, the corresponding signatures for these keys are computed and the challenge-ciphertext ct^* is computed. It uses the simulator S to create H_{pr} and the outputs of Prove on its verification keys and it uses the extractor to gain access to the secret keys of the adversary.

To simulate the random oracles H, H_2 lazily, $\mathcal{D}$ initializes two lists $\mathcal{L}, \mathcal{L}_2 = \emptyset$. $\mathcal{D}$ first computes the following auxiliary terms.

- For $i \in [n]$ randomly draw $\xi_i \leftarrow_\$ \mathbb{Z}_p$.
- Define the Lagrange polynomials

$$L'_0(x) = \prod_{j\in I} \frac{x - \xi_j}{-\xi_j} \text{ and } L'_i(x) = \frac{x}{\xi_i} \prod_{j\in I\setminus\{i\}} \frac{x - \xi_j}{\xi_i - \xi_j}$$

  for $i \in I$. That is, the L'_i are an interpolation basis for the support points $\{0\} \cup \{\xi_j \mid j \in I\}$.

- For every $i \in \bar{I}$ choose $x_i' \leftarrow \mathbb{Z}_p$ uniformly at random and set $h_{1,i} = h_1^{L_0'(\xi_i)} \cdot g_1^{x_i'}$, $h_{2,i} = h_2^{L_0'(\xi_i)} \cdot g_2^{x_i'}$.
- Choose $y \leftarrow \mathbb{Z}_p$ uniformly at random and set $A = e(v, g_2)^y / z$ and $B = g_T^y / e(h_1, w)$.
- The reduction sets $(H_0, \tau_0) \leftarrow S(0, 1^\lambda)$. It also sets a counter $ctr = 1$.

The verification keys vk_i, random-oracle queries, signature queries and the challenge ciphertext ct^* are now computed as follows.

- For every honest party with index $i \in \bar{I}$, $\mathcal{D}$ computes the verification key vk_i as $\mathsf{vk}_i = h_{2,i}$ and generates the associated proof of knowledge by $(H_{ctr}, \pi_i, \tau_{ctr}) \leftarrow S(ctr, \mathsf{vk}_i, \tau_{ctr-1}, \mathtt{YES})$, incrementing ctr after each call. Finally, we set $H_{pr} = H_{ctr}$ and make it available to $\mathcal{A}$. Due to the zero-knowledge property of the proof of knowledge, this is possible without noticeably changing the distribution of the output from either the real experiment $\mathsf{Exp}_{\mathsf{Sec}}$ or $\mathsf{Exp}'_{\mathsf{Sec}}$, except with negligible probability.
- When $\mathcal{A}$ sends (vk_i, π_i) for $i \in I$, we proceed as follows.
  - If $\mathsf{Valid}^{H_{pr}}(\mathsf{vk}_i, \pi_i) \neq 1$ for any $i \in I$ we return $\perp$.
  - Otherwise, we compute and store $\mathsf{sk}_i \leftarrow \mathcal{E}(\mathsf{vk}_i, \pi_i, Q_{\mathcal{A}})$, where $Q_{\mathcal{A}}$ denotes the queries to H_{pr} that $\mathcal{A}$ made so far.

  By extractability, extraction of the secret keys succeeds except with negligible probability. Since $\mathcal{A}$ must give a valid proof for all of its polynomially many keys, we extract all their secret keys except with negligible probability.
- Every query to H for a message $T \neq T_i$ for all $i \in SC$ (or before the challenge messages are announced) is answered as follows: If H has been queried on T before, retrieve the pair (T, α_T) from the list $\mathcal{L}$. Otherwise, choose α_T uniformly at random, add (T, α_T) to $\mathcal{L}$. Output $g_1^{\alpha_T}$. We will program H on $(T_i)_{i \in SC}$ specifically later on.
- Every query to H_2 on some input X is treated similarly: Initially, we add (vk_i, ξ_i) to $\mathcal{L}_2$ for every $i \in [n]$. If $\mathcal{L}_2$ has an entry for X, retrieve the pair (X, β_X) from the list $\mathcal{L}_2$. Otherwise, choose β_X uniformly at random, add (X, β_X) to $\mathcal{L}_2$. Output β_X.
- For every signature query of a message $T \neq T_i$ for all $i \in SC$ (or before the challenge messages are announced) for an honest party with index $i \in \bar{I}$, $\mathcal{D}$ computes the signature σ as follows. Determine $H(T)$ and retrieve the pair (T, α_T) from the list $\mathcal{L}$. Output $\sigma = h_{1,i}^{\alpha_T}$.
- $\mathcal{D}$ computes the challenge-ciphertext ct^* as follows.
  - Set $c = w$.
  - Draw randomly $\gamma \leftarrow \mathbb{Z}_p$.
  - Set $h = h_2 \cdot g_2^\gamma$.
  - Set $c_0 = w^\gamma \cdot g_2^y$.
  - For all $i \in I$ choose s_i uniformly at random and set $c_i = w^{x_i} \cdot g_2^{s_i}$.
  - For all $i \in \bar{I}$ we set $c_i = g_2^{L_0'(\xi_i) y + \sum_{j \in I} L_j'(\xi_i) \cdot s_j} \cdot w^{x_i'}$.
  - For all $j \in [\ell] \setminus SC$ set $c_j' = \frac{e(g_1, g_2)^{\alpha_{T_j} y}}{e(h_1, w)^{\alpha_{T_j}}} g_T^{m_j}$ where (T_j, α_{T_j}) is from $\mathcal{L}$.
  - For $i \in SC$ choose $\gamma_i, \delta_i \leftarrow \mathbb{Z}_p$ uniformly at random, program $H(T_i) = v^{\gamma_i} \cdot g_1^{\delta_i}$ and set $c_i' = A^{\gamma_i} \cdot B^{\delta_i} \cdot g_T^{m_i}$.

We will now show the following:

1. If $(g_1, h_1, v, g_2, h_2, w, z)$ follows the BDH distribution, i.e. $h_1 = g_1^x$, $h_2 = g_2^x$, $v = g_1^\alpha$, $w = g_2^r$ and $z = g_T^{\alpha x r}$, then $\mathcal{D}$ simulates the security experiment $\mathsf{Exp}_{\mathsf{Sec}}(\mathcal{A}, 1^\lambda)$ of SWE perfectly from the view of $\mathcal{A}$.

2. If $(g_1, h_1, v, g_2, h_2, w, z)$ follows the random distribution, i.e. $h_1 = g_1^x$, $h_2 = g_2^x$, $v = g_1^\alpha$, $w = g_2^r$ and $z = u$ for a uniformly random $u \leftarrow \mathbb{G}_T$, then $\mathcal{D}$ simulates the modified experiment $\mathsf{Exp'}_{\mathsf{Sec}}(\mathcal{A}, 1^\lambda)$ of SWE perfectly from the view of $\mathcal{A}$.

Thus, $\mathcal{A}$'s advantage in deciding between the two is at least ε and it will follow that the distinguishing advantage of $\mathcal{D}$ against BDH is at least ε.

We will first analyze the distribution of the vk_i, the signatures σ and the challenge-ciphertext components h, c_0, c and c_i.

We will first calculate the terms $h_{1,i}$ and $h_{2,i}$ for $i \in \bar{I}$. It holds that

$$\begin{aligned} h_{1,i} &= h_1^{L'_0(\xi_i)} \cdot g_1^{x'_i} = g_1^{L'_0(\xi_i)x + x'_i} = g_1^{\tilde{x}_i} \\ h_{2,i} &= h_2^{L'_0(\xi_i)} \cdot g_2^{x'_i} = g_2^{L'_0(\xi_i)x + x'_i} = g_2^{\tilde{x}_i}, \end{aligned}$$

where we set $\tilde{x}_i = L'_0(\xi_i)x + x'_i$. Note that since the x'_i are uniformly random, so are the $\tilde{x}_i$.

Hence, for the verification keys vk_i for $i \in \bar{I}$ it holds that

$$\mathsf{vk}_i = h_{2,i} = g_2^{\tilde{x}_i}.$$

Next, we consider the distribution of the signatures σ of a message T created upon a signing request for an honest key vk_i for $i \in \bar{I}$. It holds that

$$\sigma = h_{1,i}^{\alpha_T} = g_1^{\alpha_T \cdot \tilde{x}_i} = H(T)^{\tilde{x}_i}.$$

Regarding the challenge-ciphertext ct^*, let us make some definitions. We define $r_0 = y - rx$ and set f to be the (uniquely defined) polynomial of degree $t-1$ obtained by interpolating the pairs $(0, r_0), (\xi_i, s_i)_{i \in I}$. For $i \in \bar{I}$, we now set $s_i = f(\xi_i)$. Now, the following holds:

- $c = w = g_2^r$
- $h = h_2 \cdot g_2^\gamma$ is uniformly distributed
- $c_0 = w^\gamma \cdot g_2^y = g_2^{\gamma r} \cdot g_2^{y - xr + xr} = (g_2^{\gamma + x})^r \cdot g_2^{r_0} = h^r \cdot g_2^{r_0}$
- For $i \in I$ it holds that

$$c_i = w^{x_i} \cdot g_2^{s_i} = g_2^{r \cdot x_i} \cdot g_2^{s_i} = \mathsf{vk}_i^r \cdot g_2^{s_i}.$$

- For $i \in \bar{I}$ it holds that

$$\begin{aligned} c_i &= g_2^{L'_0(\xi_i) \cdot y + \sum_{j \in I} L'_j(\xi_i) s_j} \cdot w^{x'_i} \\ &= g_2^{L'_0(\xi_i) \cdot (rx + r_0) + r x'_i + \sum_{j \in I} L'_j(\xi_i) s_j} \\ &= g_2^{r(L'_0(\xi_i)x + x'_i) + L'_0(\xi_i) r_0 + \sum_{j \in I} L'_j(\xi_i) s_j} \\ &= g_2^{r(L'_0(\xi_i)x + x'_i) + f(\xi_i)} \\ &= g_2^{r\tilde{x}_i + s_i} \\ &= \mathsf{vk}_i^r \cdot g_2^{s_i}. \end{aligned}$$

Note that the s_i have the proper distribution: r_0 as well as the s_i for $i \in I$ are uniformly random and independent. Thus f is a uniformly random polynomial of degree $t-1$. Next, we consider the ciphertext components c'_j for $j \in [\ell] \setminus SC$. It holds

$$\begin{aligned} c'_j &= \frac{e(g_1, g_2)^{\alpha_{T_j} y}}{e(h_1, w)^{\alpha_{T_j}}} g_T^{m_j} \\ &= \frac{e(g_1, g_2)^{\alpha_{T_j} y}}{e(g_1^x, g_2^r)^{\alpha_{T_j}}} g_T^{m_j} \\ &= e(g_1, g_2)^{\alpha_{T_j}(y - xr)} g_T^{m_j} \\ &= e(g_1^{\alpha_{T_j}}, g_2)^{r_0} g_T^{m_j} \\ &= e(H(T_j), g_2)^{r_0} g_T^{m_j}. \end{aligned}$$

This conforms to the regular distribution.

We will finally consider the ciphertext components c'_i for $i \in SC$. In the first case, assume that $(g_1, h_1, v, g_2, h_2, w, z)$ follows the BDH distribution, i.e. $z = g_T^{\alpha x r}$. In this case, it holds that

$$A = e(v, g_2)^y / z = g_T^{\alpha(rx + r_0)} \cdot g_T^{-\alpha x r} = g_T^{\alpha r_0}$$

and

$$B = g_T^y / e(h_1, w) = g_T^{rx + r_0} \cdot g_T^{-xr} = g_T^{r_0}.$$

It follows that

$$H(T_i) = v^{\gamma_i} \cdot g_1^{\delta_i} = g_1^{\gamma_i \alpha + \delta_i} = g_1^{\alpha_i},$$

where we set $\alpha_i = \gamma_i \alpha + \delta_i$. Note that since δ_i is chosen uniformly random, α_i is distributed uniformly random.

Now, c'_i is distributed according to

$$\begin{aligned} c'_i &= A^{\gamma_i} \cdot B^{\delta_i} \cdot g_T^{m_i} \\ &= g_T^{\alpha r_0 \gamma_i} \cdot g_T^{r_0 \delta_i} \cdot g_T^{m_i} \\ &= g_T^{r_0 \cdot (\gamma_i \alpha + \delta_i)} \cdot g_T^{m_i} \\ &= g_T^{\alpha_i r_0} \cdot g_T^{m_i} \\ &= e(g_1^{\alpha_i}, g_2)^{r_0} \cdot g_T^{m_i} \\ &= e(H(T_i), g_2)^{r_0} \cdot g_T^{m_i}. \end{aligned}$$

Thus, c'_i has the same distribution as in the SWE security experiment $\mathsf{Exp}_{\mathsf{Sec}}$.

On the other hand, if $(g_1, h_1, v, g_2, h_2, w, z)$ follows the random distribution, then write z as $z = g_T^{\alpha x r + \tau}$ for a uniformly random and independent τ. Since τ is uniformly random, it holds that $\tau \neq 0$, except with negligible probability $1/p$. Thus assume in the following that $\tau \neq 0$. The terms B and $H(T_{\mathsf{ind}})$ are computed as above. The term A is now of the form

$$A = e(v, g_2)^y / z = g_T^{\alpha(rx + r_0)} \cdot g_T^{-\alpha x r - \tau} = g_T^{\alpha r_0 - \tau}.$$

Finally, the terms c'_i for $i \in SC$ are of the form:

$$\begin{aligned} c'_i &= A^{\gamma_i} \cdot B^{\delta_i} \cdot g_T^{m_i} \\ &= g_T^{r_0 \cdot (\gamma_i \alpha + \delta_i) - \tau \gamma_i} \cdot g_T^{m_i} \\ &= g_T^{r_0 \cdot (\gamma_i \alpha + \delta_i)} g_T^{-\tau \gamma} \cdot g_T^{m_i} \end{aligned}$$

Now note that since γ_i and δ_i are uniformly random and independent, $\gamma_i \alpha + \delta_i$ and $\tau \gamma_i$ are also uniformly random and independent as $\tau \neq 0$[12]. Since the term $g_T^{-\tau\gamma}$ is uniformly random and

[12] This can be seen as the matrix $\begin{pmatrix} \alpha & 1 \\ \tau & 0 \end{pmatrix}$ has full rank given that $\tau \neq 0$.

independent of all other terms, it follows that c_i' is uniformly random and thus the output is distributed as in the modified experiment $\mathsf{Exp'}_{\mathsf{Sec}}(\mathcal{A}, 1^\lambda)$ from the view of $\mathcal{A}$.

If an adversary could distinguish between the two experiments, that therefor implies our distinguisher can decide BDH with the same advantage.

Lastly, note that the output in $\mathsf{Exp'}_{\mathsf{Sec}}(\mathcal{A}, 1^\lambda)$ is independent of the challenge messages m_i^b. Consequently, in this experiment the advantage of any adversary $\mathcal{A}$ is 0. Security of SWE follows. □

Theorem 10. *Assume that the hash functions H, H_2, H_{pr} are modelled as random oracles. Then SWE' for the signature scheme Sig' is secure under the BDH assumption in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$. The security reduction is tight.*

Proof. We have previously shown Lemma 9, which allows us for the base scheme SWE to replace the ciphertext parts ct_i' output in the security experiment, by random elements for all challenge locations, as long as all T_i are distinct.

To leverage this, we first argue that we can blackbox use the outputs of an honest SWE security challenger (for known plaintext) to faithfully enact an SWE' security challenger. Note that SWE and SWE' only differ by SWE' additionally providing $a_i = c^{\alpha_i}$ and $t_i = H(T_i)^{\alpha_i}$ and having a modified structure of the ciphertext parts $c_i' = e(H(T_i), g)^{r_0\alpha_i} \cdot g_T^{m_i}$. We do this, by asking for the required $H(T_i)$ and encrypting only one challenge message $m_i = m_i^1 = m_i^0$ for each T_i used, which gives us c and ct_i' such that we can regain $e(H(T_i), g)^{r_0} = \mathsf{ct}_i'/g_T^{m_i}$. From there, we only need to pick α_i to compute the required a_i, t_i and modified c_i'. This allows us to the apply the lemma and essentially swap out $e(H(T_i), g)^{r_0}$ for a random value u_i in that construction and output $ct_i' = u_i^{\alpha_i} g_T^{m_i}$.

We notice, that if $T_i = T_j$, then the resulting ciphertext pieces ct_i' share the same u_i. From there, we use a modified BDH distribution to show that such $\mathsf{ct}_i', \mathsf{ct}_j'$ still look independent and uniformly random.

First, let us define our modified BDH assumption and show, that it in fact follows directly from BDH:

Lemma 11. *Assuming the BDH assumption holds in groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order p. Then, the following distributions are computationally close:*

$$(g_1, g_2, g_1^\alpha, g_2^\alpha, g_T^r, g_T^{r\alpha}) \approx_c (g_1, g_2, g_1^\alpha, g_2^\alpha, g_T^r, g_T^u),$$

where $\alpha, r, u \leftarrow_\$ \mathbb{Z}_p$ and $g_1 \in G_1, g_2 \in G_2, g_T = e(g_1, g_2) \in G_T$ are generators.

The reduction is tight.

Proof. We show that given a distinguisher $\mathcal{D}$ with advantage ε for the above distributions can be used to build a distinguisher against the BDH assumption with the same advantage.

Our BDH distinguisher is given a tuple $(g_1, X_1, Z, g_2, X_2, S, Y)$ which is either distributed as BDH tuple $(g_1, g_1^x, g_1^z, g_2, g_2^x, g_2^s, g_T^{zxs})$ or as random tuple $(g_1, g_1^x, g_1^z, g_2, g_2^x, g_2^s, g_T^y)$, where $x, y, z, s \leftarrow_\$ \mathbb{Z}_p$.

We input $(g_1, g_2, X_1, X_2, e(Z, S), Y)$ to the distinguisher $\mathcal{D}$ and output what they output. If our input was a BDH tuple, this is distributed as $(g_1, g_2, g_1^x, g_2^x, e(g_1, g_2)^{sz}, e(g_1, g_2)^{zxs})$. This is indentically distributed to the left-hand distribution with $\alpha = x, r = sz$ being random values from $\mathbb{Z}_p$.

If our input was a random tuple, this is distributed as $(g_1, g_2, g_1^x, g_2^x, e(g_1, g_2)^{sz}, e(g_1, g_2)^y)$. This is indentically distributed to the right-hand distribution with $\alpha = x, r = sz, u = y$ being random values from $\mathbb{Z}_p$.

Therefore, our distinguishing advantage is identical to that of $\mathcal{D}$ and we conclude, that the distributions are computationally indistinguishable if the BDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$.

□

Let us assume, that there is an adversary with advantage ε in the security game $\mathsf{Exp}_{\mathsf{Sec}}(\mathcal{A}, 1^\lambda)$ for SWE'. We define a series of hybrids and claim that they are indistinguishable from the view of $\mathcal{A}$.

$\mathcal{H}_0$ This is the real game.

$\mathcal{H}_1$ This experiment behaves exactly as the $\mathcal{H}_\prime$, except, that once we receive the challenge m_i^0, m_i^1 for $i \in SC$, a list of messages $(m_i)_{i\in[\ell]\setminus SC}$ and a list of signing reference messages $(T_i)_{i\in[\ell]}$ from the adversary, the output ct is not created as

$$\mathsf{ct} = \mathsf{SWE}'.\mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell],(m_i^b)_{i\in[\ell]}})$$

but instead:

- Let $L = \{i \in SC | \forall_{j\in SC:j<i}(T_j \neq T_i)\} \cup \{i \in [\ell] | \forall_{j\in SC\cup[i-1]:}(T_j \neq T_i)\}$ be a subset of indices, such that $(T_i)_{i\in L}$ is a set of all the distinct T_i provided by the adversary (essentially, if some T is ever used at a challenge index $i \in SC$, then the lowest index in SC with that particular T is included in L, otherwise, just the lowest overall index is chosen). We define $\phi(i) = \{j \in L | T_j = T_i\}$ to be the mapping, of the index i of a given T_i into L.
- We run the base algorithm $\mathsf{ct}' = \mathsf{SWE}.\mathsf{Enc}(1^\lambda, V, (T_i)_{i\in L}, (m_i)_{i\in L})$, where we choose $m_i = m_i^b$ for random $b \leftarrow_\$ \{0,1\}$ for all $i \in SC \cap L$.
- We parse $\mathsf{ct}' = (h, c, c_0, (c_j)_{j\in[n]}, (c_i')_{i\in L})$.
- We choose random $\alpha_i \leftarrow_\$ \mathbb{Z}_p$ for all $i \in [\ell]$ and determine $H(T_i)$. Then we can set $t_i = H(T_i)^{\alpha_i}, a_i = c^{\alpha_i}$.
- Further, we compute for $i \in L$: $u_i = (c_i')/g_T^{m_i^b}$
- Then we can set for all $i \in [\ell]$, $c_i^* = u_{\phi(i)}^{\alpha_i} g_T^{m_i^b}$.
- Output $\mathsf{ct} = (h, c, c_0, (c_j)_{j\in[n]}, (c_i^*, a_i, t_i)_{i\in L})$

$\mathcal{H}_2$ This is identical to $\mathcal{H}_1$, except we choose random $u_i \leftarrow_\$ \mathbb{G}_T$ for $i \in SC \cap L$.

$\mathcal{H}_3$ This is identical to $\mathcal{H}_1$, except we choose random $c_i^* \leftarrow_\$ \mathbb{G}_T$ for $i \in SC$.

Lemma 12. *$\mathcal{H}_0$ and $\mathcal{H}_1$ are identically distributed.*

Proof. In $\mathcal{H}_0$, the output is $\mathsf{ct} = (h, c = g_2^r, c_0 = h^r g_2^{r_0}, (c_j = \mathsf{vk}_j^r g_2^{s_j})_{j\in[n]}, (c_i' = e(t_i, g_2^{r_0}) \cdot g_t^{m_i^b}, a_i = c^{\alpha_i}, t_i = H(T_i)^{\alpha_i})_{i\in L})$ for random $h \leftarrow \mathbb{G}_2$ and $r, r_0, \alpha_i \leftarrow \mathbb{Z}_p$ and $s_j = f(H_2(\mathsf{vk}_j))$, where f is a random degreee $t-1$ polynomial with $f(0) = r_0$.

In $\mathcal{H}_1$, we receive $\mathsf{ct} = (h, c = g_2^r, c_0 = h^r g_2^{r_0}, (c_j = \mathsf{vk}_j^r g_2^{s_j})_{j\in[n]}, (c_i' = e(H(T_i), g_2^{r_0}) \cdot g_t^{m_i^b})_{i\in L}))$ for random $h \leftarrow \mathbb{G}_2$ and $r, r_0 \leftarrow \mathbb{Z}_p$ and share $s_j = f(H_2(\mathsf{vk}_j))$ as above and output $\mathsf{ct} = (h, c, c_0, (c_j)_{j\in[n]}, (c_i^*, c^{\alpha_i}, t_i = H(T_i)_i^{\alpha})_{i\in L})$. By the random choice of α_i, all parts of the distribution agree directly, except we still need to argue that $c_i^* = e(t_i, g_2^{r_0}) \cdot g_t^{m_i^b}$.

By our definition of L and ϕ, we first note, that $c_i^* = u_{\phi(i)}^{\alpha_i} g_T^{m_i^b}$ is well-defined for all $i \in [\ell]$, as we've received u_i for $i \in L$ and ϕ maps directly into L. Now, $c_i^* = ((c_{\phi(i)}')/g_T^{m_{\phi(i)}^b})^{\alpha_i} g_T^{m_i^b}$ By definition of $c_{\phi(i)}'$, this means $c_i^* = (e(H(T_{\phi(i)}), g_0^r)^{\alpha_i} g_T^{m_i^b} = (e(t_i, g_0^r)^{\alpha_i} g_T^{m_i^b}$ and is thus correctly distributed. □

Lemma 13. *$\mathcal{H}_1$ and $\mathcal{H}_2$ are indistinguishable given that the BDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$.*

Proof. Given that $\mathcal{H}_1$ exactly implements the security game for the base scheme SWE, if we were to directly output ct', thus, we can use Lemma 9 to directly replace all ct'_i for $i \in L \cap SC$ with a random element $u'_i \leftarrow \mathbb{G}_T$. Then, since ct'_i is the only place, where the input m_i is used, we can compute $u_i = u'_i / g_T^{m_i^b}$ as a randomly distributed element for which $\mathsf{ct}'_i = u_i \cdot g_T^{m_i^b}$. $\square$

Lemma 14. *$\mathcal{H}_2$ and $\mathcal{H}_3$ are computationally indistinguishable given that the BDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$.*

Proof. We reduce distinguishing these hybrids to deciding the modified BDH problem as referenced in Lemma 11 (which tightly reduces to BDH as shown).

Let us receive a tuple $(g_1, g_2, A_1, A_2, Y, Z)$, where $A_1 = g_1^\alpha, A_2 = g_2^\alpha, Y = g_T^y$ for random $y, \alpha \leftarrow_\$ \mathbb{Z}_p$ and Z is either $g_T^{y\alpha}$ or a freshly random g_T^u.

We build a challenger that behaves exactly as in $\mathcal{H}_2$, except in the way, that $H(T_i)$ for $i \in SC$ and ct are chosen.

Let the adversary's challenge be m_i^0, m_i^1 for $i \in SC$, a list of messages $(m_i)_{i \in [\ell] \setminus SC}$ and a list of signing reference messages $(T_i)_{i \in [\ell]}$. We define L and $\phi(i)$ as above. Now, first, we re-randomize our input tuple:

- We choose freshly random $v_i, w_i \leftarrow_\$ \mathbb{Z}_p$ for $i \in [\ell]$ and $\tau_i, y'_i \leftarrow_\$ \mathbb{Z}_p$ for $i \in L$. Further set $\tau_i = \tau_{\phi(i)}, y'_i = y'_{\phi(i)}$ for $i \in [\ell] \setminus L$.
- For $i \in L$: $g_{1,i} = g_1^{\tau_i}$, $Y_i = Y^{y'_i}$ and
- For $i \in [\ell]$: $A_{1,i} = (g_1^{v_i} A_1^{w_i})^{\tau_i}$, $A_{2,i} = g_2^{v_i} A_2^{w_i}$, $Z_i = (Y^{v_i} Z^{w_i})^{y'_i}$.

We choose $H(T_i) = g_{1,\phi(i)}$. By definition of ϕ, this ensures that $H(T_i) = H(T_j)$ if $T_i = T_j$, thus making this a well-defined assignment. Further, since we choose $g_{1,\phi(i)} = g_1^{\tau_{\phi(i)}}$, for freshly random $\tau_{\phi(i)}$ for each distinct $T_{\phi(i)}$, $\phi(i) \in L$, this ensures that the output hashes are distributed randomly as in both experiment H_2 and H_2.

The output ct is generated as follows:

- We run the base algorithm $\mathsf{ct}' = \mathsf{SWE.Enc}(1^\lambda, V, (T_i)_{i \in L}, (m_i)_{i \in L})$, where we choose $m_i = m_i^b$ for random $b \leftarrow_\$ \{0,1\}$ for all $i \in SC \cap L$ and gain $\mathsf{ct}' = (h, c = g_2^r, c_0, (c_j)_{j \in [n]}, (c'_i)_{i \in L})$.
- For $i \in [\ell] \setminus SC$, we pick random $\alpha_i \leftarrow_\$ \mathbb{Z}_p$ and set $a_i = c^{\alpha_i}, t_i = H(T_i)^{\alpha_i}$. For all $i \in [\ell] \cap SC$: we set $t_i = A_{1,i}, a_i = A_{2,i}^r$
- Further we set for $i \in L \setminus SC$ $u_i = (c'_i)/g_T^{m_i^b}$ and for $i \in L \cap SC$: $u_i = Y_i$.
- Lastly, we set for $i \in [\ell] \setminus SC$ $c_i^* = u_{\phi(i)}^{\alpha_i} g_T^{m_i^b}$ and for $i \in [\ell] \cap SC$: $c_i^* = Z_i \cdot g_T^{m_i^b}$.
- Output $\mathsf{ct} = (h, c, c_o, (c_j)_{j \in [n]}, (c_i^*, a_i, t_i)_{i \in L})$

Note first: the outputs for indices $i \in [\ell] \setminus SC$ are chosen exactly as in H_2, as these are identically generated also in H_3, there is nothing to show.

Now let us discuss outputs a_i, t_i for $i \in SC$:

$$
\begin{aligned}
t_i &= A_{1,i} = (g_1^{v_i} A_1^{w_i})^{\tau_i} = (g_1^{\tau_i})^{v_i + w_i \alpha} = H(T_i)^{\tilde{\alpha}_i} \\
a_i &= A_{2,i}^r = (g_2^{v_i} A_2^{w_i})^r = (g_2^r)^{v_i + w_i \alpha} = c^{\tilde{\alpha}_i}
\end{aligned}
$$

Where $\tilde{\alpha}_i$ is defined as $v_i + w_i \alpha$.

The $u_i = Y_i$ for $i \in L \cap SC$ are distributed as $u_i = Y^{y'_i}$ for a fresh y'_i for each $i \in L \cap SC$ and therefore randomly distributed as required in H_2. (These inputs aren't actually output and not needed in H_3 at all.)

Now let us regard the outputs c_i^* for $i \in SC$:

- If $Z = g_T^{y\alpha}$, our output is identically distributed to H_2:
  For all $i \in SC$,

$$c_i^*/g_T^{m_i^b} = Z_i = (Y^{v_i} Z^{w_i})^{y_i'} = (g_T^{yv_i} g_T^{y\alpha w_i})^{y_i'}$$
$$= ((g_T^{yy_i'})^{v_i+\alpha w_i}) = (Y^{y_i'})^{\tilde{\alpha}_i} = (Y^{y'_{\phi(i)}})^{\tilde{\alpha}_i} = u_{\phi_i}^{\tilde{\alpha}_i}$$

  This is the same $\tilde{\alpha}_i$ as in a_i, t_i and thus, the output distribution is equivalent to H_2, given that all the $\tilde{\alpha}_i = v_i + w_i\alpha$ are freshly random for each $i \in [\ell]$, given that the v_i, w_i are.
- If we got a random $Z = g_T^u$, our output is identically distributed to H_3:
  We can write $u = y\alpha + \delta$, where δ is a uniquely determined random value derived from Z. Now, for all $i \in SC$,

$$c_i^*/g_T^{m_i^b} = Z_i = (Y^{v_i} Z^{w_i})^{y_i'} = (g_T^{yv_i} g_T^{(y\alpha+\delta)w_i})^{y_i'}$$
$$= ((g_T^{yy_i'})^{v_i+w_i\alpha} \cdot g_T^{\delta w_i y_i'}) = (u_{\phi_i}^{\tilde{\alpha}_i} \cdot g_T^{\delta_i})$$

  Where we define $\delta_i = \delta w_i y_i'$. Note that given δ, y_i' for $i \in L$ are freshly random values. Thus, the probability of $\delta y_i' = 0$ is negligible. Then, given we chose for $i \in [\ell]$ fresh v_i, w_i, the derived values $\tilde{\alpha}_i, \delta_i$ for $i \in SC \subseteq [\ell]$ are also two independent randomly distributed values.[13] Thus, as δ_i is independent from all other outputs, we receive, that $c_i^* = u_{\phi_i}^{\tilde{\alpha}_i} \cdot g_T^{\delta_i} g_T^{m_i^b}$ is actually randomly distributed in $\mathbb{G}_T$.

As a result, if an attacker could distinguish these hybrids, we could decide the modified BDH distribution from Lemma 11 with essentially the same advantage.

$\square$

In hybrid H_3, clearly, the advantage of an attacker must be 0, as the outputs do not depend on the challenge messages m_i^b for $i \in SC$ anymore. This concludes the proof. Further we note that all reductions were tight.

$\square$

7 A Compatibility Layer for Proof Systems

We will now construct a compatibility layer for our SWE scheme and common proof systems.

The high-level idea of this compatibility layer is to attach a proof-system-friendly commitment to a ciphertext and provide an efficient NIZK proof guaranteeing that ciphertext and commitment encrypt the same value. We can then use efficient and readily available proof systems such as Bulletproofs [13] to establish additional properties about the encrypted message.

We focus on the enhanced SWE', as it is slightly more complex. All the proofs work analogously for the base SWE and we give a few comments, where appropriate on how they need to be modified to apply to the base scheme.

7.1 Well-formedness Proofs

In this section we will provide a proof system to efficiently prove that a given SWE' ciphertext is decryptable. More precisely, this proof system will ensure that the SWE' scheme is committing in the sense that regardless of which committee-members contribute to the aggregated signature, the decrypted message is always the same. This proof system will not yet ensure that the

[13] This can again be seen as the matrix $\begin{pmatrix} \alpha & 1 \\ y_i'\delta & 0 \end{pmatrix}$ has full rank given that $y_i'\delta \neq 0$.

message m_i are in the correct range, though. This will be ensured using the proof systems in Sections 7.2 and 7.3.

We will provide a proof system to certify that ciphertexts generated by $\mathsf{SWE}'.\mathsf{Enc}$ are well-formed. That is, we define a NIZK proof $(\mathsf{P}_1, \mathsf{V}_1)$ for the language $\mathcal{L}_1$ defined by the relation

$$x = (\mathsf{ct}, (\mathsf{vk}_j)_j, (T_i)_i), w = ((m_i)_i, r_1)) \in \mathcal{R}_1 \Leftrightarrow$$
$$\mathsf{ct} = \mathsf{SWE}'.\mathsf{Enc}(1^\lambda, (\mathsf{vk}_j)_j, (T_i)_i, (m_i)_i; r_1).$$

The ciphertexts produced by $\mathsf{SWE}'.\mathsf{Enc}$ are of the form $\mathsf{ct} = (h, c, c_0, (c_j)_{j\in[n]}, (c_i', a_i, t_i)_{i\in[\ell]})$ as follows:

$$\begin{aligned} c &= g_2^r \\ c_0 &= h^r \cdot g_2^{r_0} \\ c_j &= \mathsf{vk}_j^r \cdot g_2^{s_j} \text{ for } j = 1, \dots, n \\ a_i &= c^{\alpha_i} \text{ for } i = 1, \dots, \ell \\ t_i &= H(T_i)^{\alpha_i} \text{ for } i = 1, \dots, \ell \\ c_i' &= e(t_i, g_2)^{r_0} \cdot g_T^{m_i} \text{ for } i = 1, \dots, \ell \end{aligned}$$

where $(s_1, \dots, s_n)$ is the vector of shares of r_0 under Shamir's secret sharing scheme. Since $\mathbf{s} = (r_0, s_1, \dots, s_n)$ is a codeword of the Reed-Solomon codes $\mathsf{RS}[\mathbb{Z}_q, n+1, t]$, it holds that $\mathbf{H} \cdot \mathbf{s} = 0$, where $\mathbf{H} \in \mathbb{Z}_p^{k\times(n+1)}$ is the parity-check matrix of $\mathsf{RS}[\mathbb{Z}_q, n+1, t]$.

To prove well-formedness of such a ciphertext, we proceed as follows. Let H' and H'' be hash-functions (modelled as a random oracle).

$\mathsf{P}_1(\mathsf{ct}, (\mathsf{vk}_j)_j, r)$: The prover proceeds as follows, requiring only $r \in \mathbb{Z}_p$ as witness.

- Compute $\mathbf{v} = H'(\mathsf{ct}) \in \mathbb{Z}_p^k$ and set $\mathbf{w}^\top = \mathbf{v}^\top \cdot \mathbf{H}$.
- Parse $\mathbf{w} = (w_0, w_1, \dots, w_n)$.
- Set $c^* = c_0^{w_0} \cdot \prod_{j=1}^n c_j^{w_j}$.
- Set $g^* = h^{w_0} \cdot \prod_{j=1}^n \mathsf{vk}_j^{w_j}$.
- Choose $y \leftarrow \mathbb{Z}_p$ uniformly at random and set $f = g_2^y$ and $f^* = (g^*)^y$.
- Compute $\alpha = H''(g_2, c, g^*, c^*, f, f^*)$.
- Compute $z = y + \alpha r$.
- Output $\pi = (f, f^*, z)$.

$\mathsf{V}_1(\mathsf{ct}, (\mathsf{vk}_j)_j, \pi)$: To verify a proof $\pi = (f, f^*, z)$, proceed as follows.

- Check if $e(t_i, c) = e(H(T_i), a_i)$ for all $i \in [\ell]$. Otherwise, output 0.
- Compute $\mathbf{v} = H'(\mathsf{ct})$ and set $\mathbf{w} = \mathbf{v}^\top \cdot \mathbf{H}$. Parse $\mathbf{w} = (w_0, w_1, \dots, w_n)$.
- Set $c^* = c_0^{w_0} \cdot \prod_{j=1}^n c_j^{w_j}$.
- Set $g^* = h^{w_0} \cdot \prod_{j=1}^n \mathsf{vk}_j^{w_j}$.
- Compute $\alpha = H''(g_2, c, g^*, c^*, f, f^*)$.
- Check if $(c^*)^\alpha \cdot (f^*) = (g^*)^z$ and $c^\alpha \cdot f = g_2^z$, if so output 1, otherwise 0.

For SWE, we have essentially the same scheme, but omit a_i, t_i, as all α_i are set to 1. The proof system works exactly the same, except that we omit the first check of $e(t_i, c) = e(H(T_i), a_i)$ in verification, which is made to ensure that $(c, a_i), (H(T_i), t_i)$ share the same dlog relationship to allow decryption.

Theorem 15. *(P_1, V_1) is a NIZK proof system for relation $\mathcal{R}_1$ assuming H', H'' are modelled as random oracles. Concretely, if $x = (\mathsf{ct}, (\mathsf{vk}_j)_j) \notin \mathcal{L}_1$ and P_1 makes at most q queries to either H' or H'', then V_1 rejects x, except with negligible probability q/p.*

Proof. We argue Completeness, Soundness and Zero-Knowledge of this scheme.

Completeness Completeness of this proof system follows routinely.

For all $i \in [\ell]$, $e(t_i, c) = e(H(T_i)^{\alpha_i}, c) = e(H(T_i), c^{\alpha_i}) = e(H(T_i), a_i)$ holds by construction.

We observe that the verifier constructs the same values as the prover for $\mathbf{v}, \mathbf{w}, c^*, g^*, \alpha$. Now, assuming the input was such that $\mathsf{ct} = \mathsf{SWE}'.\mathsf{Enc}(1^\lambda, (\mathsf{vk}_j)_j, \cdot, \cdot; r_1)$, where r_1 contains r as the random exponent used in ciphertext part c, then it holds

$$\begin{aligned}(c^*)^\alpha \cdot (f^*) &= (c_0^{w_0} \cdot \prod_{j=1}^{n} c_j^{w_j})^\alpha \cdot (g^*)^y \\ &= ((h^r \cdot g_2^{r_0})^{w_0} \cdot \prod_{j=1}^{n} (\mathsf{vk}_j^r \cdot g_2^{s_j})^{w_j})^\alpha \cdot (g^*)^y \\ &= g_2^{r_0 w_0 \alpha} \prod_{j=1}^{n} g_2^{s_j w_j \alpha} \cdot (g^*)^{y + r\alpha} = (g^*)^z\end{aligned}$$

since $w^\top (r_0, s_1, \ldots, s_n) = \mathbf{v}^\top \cdot \mathbf{H}(r_0, s_1, \ldots, s_n) = 0$ as discussed above. The other equation checked in P_1 can be shown to hold analogously.

Soundness First note the following. A ciphertext ct is well-formed if and only if

1. $\mathbf{s} = (r_0, s_1, \ldots, s_n) \in \mathsf{RS}[\mathbb{Z}_q, n+1, t]$ and
2. $(c, a_i), (H(T_i), t_i)$ share the same dlog relationship.

(2) is directly implied by the check $e(t_i, c) = e(H(T_i), a_i)$. We focus on the fist condition: It is exactly fulfilled if $\mathbf{Hs} = 0$. Thus assume that ct is not a valid ciphertext, i.e. it holds that $\mathbf{Hs} \neq 0$. Say that a challenge $\mathbf{v}$ is *bad*, if it holds that $\mathbf{v}^\top \mathbf{Hs} = 0$. Since $\mathbf{v}$ is chosen uniformly random and $\mathbf{Hs} \neq 0$ it holds that

$$\Pr[\mathbf{v} \text{ bad}] = \Pr[\mathbf{w}^\top \mathbf{s} = 0] = \Pr[\mathbf{v}^\top (\mathbf{Hs}) = 0] = \frac{1}{p},$$

which is negligible over the random choice of $\mathbf{v}$. Note that it holds that

$$c^* = c_0^{w_0} \prod_{j=1}^{n} c_j^{w_j} = h^{r w_0} \cdot g_2^{r_0 w_0} \prod_{j=1}^{n} (\mathsf{vk}_j^r \cdot g_2^{s_j})^{w_j} = (g^*)^r \cdot g_2^{\mathbf{w}^\top \mathbf{s}}.$$

Now fix a $\mathbf{v}$ which is not bad, i.e. it holds that $\mathbf{w}^\top \mathbf{s} \neq 0$. Then it holds that $(g_2, c = g_2^r)$ and $(g^*, c^* = (g^*)^r \cdot g_T^{\mathbf{w}^\top \mathbf{s}})$ do not satisfy the same discrete logarithm relation, in other words the vectors $(1, \log_{g_2}(c))$ and $(1, \log_{g^*}(c^*))$ are linearly independent. But this means that also $(1, 1)$ and $(\log_{g_2}(c), \log_{g^*}(c^*))$ are linearly independent.

Now fix any f and f^* which may depend on $\mathbf{v}$. Say that an α is *bad* if there exists a $z \in \mathbb{Z}_p$ such that

$$\begin{aligned}c^\alpha \cdot f &= g_2^z \\ (c^*)^\alpha \cdot f^* &= (g^*)^z.\end{aligned}$$

Now observe that α is bad if and only if

$$\alpha(\log_{g_2}(c), \log_{g^*}(c^*)) + (\log_{g_2}(f), \log_{g^*}(f^*)) \in \mathsf{Span}((1, 1)),$$

which is equivalent to

$$\alpha(\log_{g_2}(c), \log_{g^*}(c^*)) \in -(\log_{g_2}(f), \log_{g^*}(f^*)) + \mathsf{Span}((1, 1)).$$

As $(1,1)$ and $(\log_{g_2}(c), \log_{g^*}(c^*))$ are linearly independent, $\alpha(\log_{g_2}(c), \log_{g^*}(c^*))$ only lands in the affine subspace $-(\log_{g_2}(f), \log_{g^*}(f^*)) + \mathsf{Span}((1,1))$ only with negligible probability $1/p$ over the choice of α. It follows that $\Pr[\alpha \text{ bad}] \leq 1/p$.

Now note that V_1 rejects if neither $\mathbf{v}$ nor α is bad. Now assume that the adversary makes q_1 queries to H' and q_2 queries to H''. By a union bound we conclude the the probability that one of the queries to H' results in a bad $\mathbf{v}$ is at most q_1/p. Furthermore, also by a union bound the probability that one of the queries to H'' results in a bad α is at most q_2/p. We conclude with a union bound that there are no bad queries to H' or H'', except with probability $(q_1+q_2)/p$, which is negligible.

Zero-Knowledge To show that this proof-system is zero-knowledge, we can routinely make use of the fact that the underlying Schnorr-like proof-system for equality of discrete logarithms (see e.g.[16]) is zero-knowledge.

Specifically, our simulator can chooses z and α uniformly at random and sets $f = g_2^z \cdot c^{-\alpha}$ and $f^* = (g^*)^z \cdot (c^*)^{-\alpha}$ and programs the random oracle H'' to output α on the query $(g_2, c, g^*, c^*, f, f^*)$.

It follows routinely that given a YES-instance $(\mathsf{ct}, (\mathsf{vk}_j)_j)$ the simulation is perfect unless the verifier makes makes a hash query $(g_2, c, g^*, c^*, f, f^*)$ to H'' that the simulator would make, thus prohibiting the simulator to program H'' accordingly. However, since f is distributed uniformly random, this happens only with negligible probability $1/p$. Thus we have established the zero-knowledge property.

□

7.2 Proofs of Plaintext Equality

First observe the following: A ciphertext $\mathsf{ct} = (h, c, c_0, (c_j)_j, (c'_i, a_i, t_i)_i)$ is a statistically binding commitment to a message $\mathbf{m} = (m_i)_i$, even if we drop the c_j, a_i. In fact, the only purpose of the c_j and a_i is to facilitate decryption. If we fix h and the $h_{T,i} = e(t_i, g_2)$, then $\mathsf{com}(\mathbf{m}; r, r_0) = (c = g_2^r, c_0 = h^r \cdot g_2^{r_0}, (c'_i = h_{T,i}^{r_0} \cdot g_T^{m_i})_i)$ is in fact a homomorphic commitment scheme[14] and it holds that

$$\mathsf{com}(\mathbf{m}; r, r_0)^\alpha \cdot \mathsf{com}(\mathbf{m}'; r', r'_0)^\beta = \mathsf{com}(\alpha\mathbf{m} + \beta\mathbf{m}'; \alpha r + \beta r', \alpha r_0 + \beta r'_0),$$

where the exponentiation is component-wise.

Homomorphic operations for this commitment scheme are relatively expensive, as the c'_i components reside in the target group $\mathbb{G}_T$. To enable efficient proofs of statements over the commited values $\mathbf{m}$, we will leverage a second commitment scheme COM and provide a highly efficient cross-scheme equality proof for two commitments $\mathsf{com}(\mathbf{m})$ and $\mathsf{COM}(\mathbf{m})$. Furthermore, we will choose the commitment scheme COM to be proof-system friendly. Thus, the natural choice for COM is a Pedersen commitment in a cryptographic group $\mathbb{G}$ of order p. For simplicity, we may choose e.g. $\mathbb{G} = \mathbb{G}_1$, but $\mathbb{G}$ could in fact be any group of order p. Thus, let $g, h_1, \ldots, h_\ell \leftarrow \mathbb{G}$ be public but randomly chosen group elements from the group $\mathbb{G}$. Then $\mathsf{COM}(\mathbf{m})$ is computed by $\mathsf{COM}(\mathbf{m}; \rho) = g^\rho \cdot \prod_{i=1}^{\ell} h_i^{m_i}$ for a uniformly random $\rho \leftarrow \mathbb{Z}_p$. This commitment is computationally binding under the discrete logarithm assumption in $\mathbb{G}$.

We will now provide a non-interactive zero-knowledge proof of knowledge $(\mathsf{P}_2, \mathsf{V}_2)$ for the following relation $\mathcal{R}_2$. For given $\mathsf{crs} = (h \in \mathbb{G}_2,\ h_{T,1}, \ldots, h_{T,\ell} \in \mathbb{G}_T)$ and $\mathsf{CRS} = (g, h_1, \ldots, h_\ell \in$

[14]The intuition to show the binding property is as follows: If there are $r, r_0, \mathbf{m}, r', r'_0, \mathbf{m}'$ such that $\mathsf{com}(\mathbf{m}; r, r_0) = \mathsf{com}(\mathbf{m}'; r', r'_0)$, then as g_2 is a generator and $c = g_2^r$, $r = r'$. Then, h^r becomes fixed and from $c_0 = h^r \cdot g_2^{r_0}$, we conclude $r_0 = r'_0$. The same holds for all m_i individually as g_T is a generator.

$\mathbb{G}$) we want to establish that for c and C it holds that $\mathsf{c} = \mathsf{com}(\mathbf{m}; r, r_0)$ and $\mathsf{C} = \mathsf{COM}(\mathbf{m}; \rho)$ for some $\mathbf{m} \in \mathbb{Z}_p^\ell$ and $r, r_0, \rho \in \mathbb{Z}_p$.

Let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a hash-function, which will be modelled as a random oracle.

$\mathsf{P}_2((\mathsf{crs}, \mathsf{CRS}, \mathsf{c}, \mathsf{C}), (\mathbf{m}, r, r_0, \rho))$:

- Choose $\mathbf{u} \leftarrow \mathbb{Z}_p^\ell$ uniformly at random.
- Choose $r_0', r', \rho' \leftarrow \mathbb{Z}_p$ uniformly at random.
- Compute $\mathsf{c}' = \mathsf{com}_{\mathsf{crs}}(\mathbf{u}; r', r_0')$ and $\mathsf{C}' = \mathsf{COM}_{\mathsf{CRS}}(\mathbf{u}; \rho')$.
- Compute $\alpha = H(\mathsf{crs}, \mathsf{CRS}, \mathsf{c}, \mathsf{C}, \mathsf{c}', \mathsf{C}')$.
- Compute $\tilde{\mathbf{m}} = \mathbf{u} + \alpha\mathbf{m}$, $\tilde{r} = r' + \alpha r$, $\tilde{r}_0 = r_0' + \alpha.r_0$ and $\tilde{\rho} = \rho' + \alpha\rho$.
- Output $\pi = (\mathsf{c}', \mathsf{C}', \tilde{\mathbf{m}}, \tilde{r}, \tilde{r}_0, \tilde{\rho})$.

$\mathsf{V}_2((\mathsf{crs}, \mathsf{CRS}, \mathsf{c}, \mathsf{C}), \pi = (\mathsf{c}', \mathsf{C}', \tilde{\mathbf{m}}, \tilde{r}, \tilde{r}_0, \tilde{\rho}))$:

- Compute $\alpha = H(\mathsf{crs}, \mathsf{CRS}, \mathsf{c}, \mathsf{C}, \mathsf{c}', \mathsf{C}')$.
- Check if $\mathsf{c}' \cdot \mathsf{c}^\alpha \stackrel{?}{=} \mathsf{com}_{\mathsf{crs}}(\tilde{\mathbf{m}}; \tilde{r}, \tilde{r}_0)$ and $\mathsf{C}' \cdot \mathsf{C}^\alpha \stackrel{?}{=} \mathsf{COM}_{\mathsf{CRS}}(\tilde{\mathbf{m}}; \tilde{\rho})$, if so output 1, otherwise 0.

We note that we can mount the same argument and run the exact same proof system for SWE, by setting $h_{T,i} = e(H(T_i), g_2))$ instead, as this results in the exact same committment scheme $\mathsf{com}(\mathbf{m}; r, r_0) = (c = g_2^r, c_0 = h^r \cdot g_2^{r_0}, (c_i' = h_{T,i}^{r_0} \cdot g_T^{m_i})_i)$ being a subset of SWE ciphertexts.

Theorem 16. *(P_2, V_2) is a NIZK proof of knowledge for relation $\mathcal{R}_2$, assuming H is modelled as random oracle.*

Proof. We argue the properties of completeness, proof of knowledge and zero-knowledge:

Completeness Completeness of this proof system follows routinely. On an honestly generated input $((\mathsf{crs}, \mathsf{CRS}, \mathsf{c}, \mathsf{C}), \pi = \mathsf{P}_2((\mathsf{crs}, \mathsf{CRS}, \mathsf{c}, \mathsf{C}), (\mathbf{m}, r, r_0, \rho)))$ to V_2 we can see that $\mathsf{c}' \cdot \mathsf{c}^\alpha = \mathsf{com}_{\mathsf{crs}}(\mathbf{u}; r', r_0') \cdot \mathsf{com}_{\mathsf{crs}}(\mathbf{m}; r, r_0)^\alpha = \mathsf{com}_{\mathsf{crs}}(\tilde{\mathbf{m}}; \tilde{r}, \tilde{r}_0)$ by our definitions of $\tilde{\mathbf{m}}, \tilde{r}, \tilde{r}_0$ and the underlying scheme being homomorphic. The same holds for $\mathsf{C}' \cdot \mathsf{C}^\alpha \stackrel{?}{=} \mathsf{COM}_{\mathsf{CRS}}(\tilde{\mathbf{m}}; \tilde{\rho})$, so on an honest input, P_2 always outputs 1.

Proof of Knowledge We will now argue that $(\mathsf{P}_2, \mathsf{V}_2)$ is a proof of knowledge.

Let CRS be a given common-references string for COM. Fix a statement $(\mathsf{crs}, \mathsf{CRS}, \mathsf{c}, \mathsf{C})$ and let P_2^* be a malicious prover.

In order to use Lemma 1 to construct a knowledge extractor, let $\mathcal{A}$ be an algorithm which first runs P_2^* to produce a candidate proof π and then uses V_2 to verify π, if it accepts $\mathcal{A}$ outputs 1 and auxiliary information π and a response α for the H-query made by P_2^*.

Now, by Lemma 1 there exists a forking algorithm $\mathsf{F}_{\mathsf{P}_2^*}$ with expected runtime only polynomially larger than that of P_2^* which, if a run of P_2^* produces a verifying proof π, produces a fork π, π'.

Our knowledge extractor $\mathcal{E}$ is now given as follow.

- Run F_P, and if it outputs 0, also output 0.
- Otherwise, if $\mathsf{F}_{\mathsf{P}_2^*}$ outputs a fork (π, α) and (π', α') proceed as follows.
- Parse $\pi = (\mathsf{c}', \mathsf{C}', \tilde{\mathbf{m}}, \tilde{r}, \tilde{r}_0, \tilde{\rho})$ and $\pi' = (\hat{\mathsf{c}}, \hat{\mathsf{C}}, \hat{\mathbf{m}}, \hat{r}, \hat{r}_0, \hat{\rho})$.
- Compute $\bar{\mathbf{m}} = (\tilde{\mathbf{m}} - \hat{\mathbf{m}})/(\alpha - \alpha')$, $\bar{r} = (\tilde{r} - \hat{r})/(\alpha - \alpha')$, $\bar{r}_0 = (\tilde{r}_0 - \hat{r}_0)/(\alpha - \alpha')$ and $\bar{\rho} = (\tilde{\rho} - \hat{\rho})/(\alpha - \alpha')$ and output $(\bar{\mathbf{m}}, \bar{r}, \bar{r_0}, \bar{\rho})$.

In case the last step is reached, it holds that

$$\mathsf{C}' \cdot \mathsf{C}^{\alpha} = \mathsf{COM}_{\mathsf{CRS}}(\tilde{\mathbf{m}}; \tilde{\rho}) \text{ and } \mathsf{C}' \cdot \mathsf{C}^{\alpha'} = \mathsf{COM}_{\mathsf{CRS}}(\hat{\mathbf{m}}; \hat{\rho}),$$

from which we can conclude that

$$\mathsf{C}^{\alpha - \alpha'} = \mathsf{COM}_{\mathsf{CRS}}(\tilde{\mathbf{m}} - \hat{\mathbf{m}}; \tilde{\rho} - \hat{\rho}),$$

and therefore

$$\mathsf{C} = \mathsf{COM}_{\mathsf{CRS}}((\tilde{\mathbf{m}} - \hat{\mathbf{m}})/(\alpha - \alpha'); (\tilde{\rho} - \hat{\rho})/(\alpha - \alpha')) = \mathsf{COM}_{\mathsf{CRS}}(\bar{\mathbf{m}}; \bar{\rho}).$$

An analogous argument can be mounted to show that $\mathsf{c} = \mathsf{com}_{\mathsf{crs}}(\bar{\mathbf{m}}; \bar{r}, \bar{r}_0)$. This means $\mathcal{E}$ is a PPT algorithm with extraction probability p', thus our knowledge extractor $\mathcal{E}$ is efficient and correct.

Zero-Knowledge To argue that $(\mathsf{P}_2, \mathsf{V}_2)$ is zero-knowledge, we construct a simulator $\mathcal{S}$ which, given a statement $(\mathsf{crs}, \mathsf{CRS}, \mathsf{c}, \mathsf{C})$, chooses a uniformly random $\tilde{\mathbf{m}} \leftarrow \mathbb{Z}_p^{\ell}$, and uniformly random $\tilde{r}, \tilde{r}_0, \rho \leftarrow \mathbb{Z}_p$ as well as a uniformly random $\alpha \leftarrow \mathbb{Z}_p$ and sets $\mathsf{c}' = \mathsf{com}_{\mathsf{crs}}(\tilde{\mathbf{m}}; \tilde{r}, \tilde{r}_0) \cdot \mathsf{c}^{-\alpha}$ and $\mathsf{C}' = \mathsf{COM}_{\mathsf{CRS}}(\tilde{\mathbf{m}}; \tilde{\rho}) \cdot \mathsf{C}^{-\alpha}$. Further, the simulator $\mathcal{S}$ programs the random oracle H to output α on input $(\mathsf{crs}, \mathsf{CRS}, \mathsf{c}, \mathsf{C}, \mathsf{c}', \mathsf{C}')$. The simulated proof π is given by $\pi = (\mathsf{c}', \mathsf{C}', \tilde{\mathbf{m}}, \tilde{r}, \tilde{r}_0, \tilde{\rho})$.

It follows routinely that the simulation is perfect, unless $\mathcal{A}$ queries H on $(\mathsf{crs}, \mathsf{CRS}, \mathsf{c}, \mathsf{C}, \mathsf{c}', \mathsf{C}')$ before obtaining the proof π. But this only happens with negligible probability as the commitments c' and C' are freshly chosen by $\mathcal{S}$. □

7.3 Putting Everything Together: Verifiable SWE

We will now briefly discuss how we can extend the SWE' scheme constructed in Section 6.1 with the proof systems of this section and an efficient proof system (e.g. Bulletproofs [13]) to obtain an efficient *verifiable* SWE scheme, i.e. an SWE for which we can efficiently prove statements about the encrypted messages.

Let a relation $\mathcal{R}$ for messages m and witnesses w be given and $\mathcal{L}$ be its induced language. We say that a vector $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathcal{L}$, if $\sum_{i \in [\ell]} 2^{(i-1)k} m_i \in \mathcal{L}$.

Let $(\mathsf{P}_1, \mathsf{V}_1)$ be the proof system for well-formedness constructed in Section 7.1, let $(\mathsf{P}_2, \mathsf{V}_2)$ be the proof system for plaintext equality constructed in Section 7.2, and finally let $(\mathsf{P}_3, \mathsf{V}_3)$ be a proof system which asserts for a given $\mathsf{C} = \mathsf{COM}_{\mathsf{CRS}}(\mathbf{m}; \rho)$ that $\mathbf{m} \in \mathcal{L}$. In the following, let w be an auxiliary witness for $\mathbf{m} \in \mathcal{L}$, i.e. P_3 takes as input a commitment $\mathsf{C}(\mathbf{m}; \rho)$ and a witness ρ, w. To ensure efficient decryption even for maliciously generated ciphertexts, the language $\mathcal{L}$ at the bare minimum must enforce that each component m_i is in the appropriate range, i.e. $m_i \in \{0, \ldots, 2^k - 1\}$ for a small integer k. This will guarantee correctness of efficient decryption with baby-step giant-step. Such efficient range-proofs are provided by the Bulletproofs proof system [13].

Lemma 17 ([13], Section 4.2)**.** *There exists a zero-knowledge range proof* (P, V) *for the commitment scheme* COM *where the proofs* π *consist of* $4k\ell + 4$ *group elements and* 5 $\mathbb{Z}_p$ *elements.*

In the following, we assume that the commitment scheme COM takes the same common reference string CRS as provided for $(\mathsf{P}_3, \mathsf{V}_3)$. In the following we describe $\mathsf{Prove}, \mathsf{Vrfy}$ that extend SWE' with $\mathsf{Enc}, \mathsf{Dec}$ as described in 6.1 to be a verifiable SWE scheme.

We require the random coins r' used in encryption to be input to the proof, and then extract the subset of random coins (r, r_0) as required by $(\mathsf{P}_1, \mathsf{V}_1)$ and $(\mathsf{P}_2, \mathsf{V}_2)$. Note that executing encryption and the proof in one instance is more efficient and preferred in practice.

$\mathsf{SWE}'.\mathsf{Prove}(\mathsf{CRS}, (\mathsf{vk}_j)_{j\in[n]}, (T_i)_{i\in[\ell]}, \mathsf{ct}, (m_i)_{i\in[\ell]}, w, r')$:

- Pick necessary random coins r, r_0 from r'.
- Parse $\mathsf{ct} = (h, c, c_0, (c_j)_{j\in[n]}, (c'_i)_{i\in[\ell]})$
- Choose $\rho \leftarrow \mathbb{Z}_p$ uniformly at random
- Compute $\mathsf{C} = \mathsf{COM}_{\mathsf{CRS}}((m_i)_{i\in[\ell]}; \rho)$.
- Compute $\pi_1 = \mathsf{P}_1(\mathsf{ct}, r')$.
- Compute $\pi_2 = \mathsf{P}_2(\mathsf{crs} = (h, (t_i)_{i\in[\ell]}), \mathsf{CRS}, (c, c_0, (c'_i)_{i\in[\ell]}), \mathsf{C}), ((m_i)_{i\in[\ell]}, r, r_0, \rho))$.
- Compute $\pi_3 = \mathsf{P}_3(\mathsf{C}, (\rho, w))$.
- Output $\pi = (\mathsf{C}, \pi_1, \pi_2, \pi_3)$.

$\mathsf{SWE}'.\mathsf{Vrfy}(\mathsf{CRS}, (\mathsf{vk}_j)_{j\in[n]}, (T_i)_{i\in[\ell]}, \mathsf{ct}, \pi)$:

- Output 1 if $\mathsf{V}_1(\mathsf{ct}, \pi_1) = 1$, $\mathsf{V}_2((\mathsf{ct}, \mathsf{C}, (\mathsf{vk}_j)_{j\in[n]}, (T_i)_{i\in[\ell]}), \pi_2) = 1$ and $\mathsf{V}_3(\mathsf{CRS}, \mathsf{C}, \pi_3) = 1$, otherwise output 0.

Theorem 18. $\mathsf{SWE}'.\mathsf{Prove}, \mathsf{SWE}'.\mathsf{Vrfy}$ *extend* SWE' *to be a verifiable SWE.*

Proof. We need to show that $\mathsf{Prove}, \mathsf{Vrfy}$ are a NIZK proof system for a language given by the induced relation $\mathcal{R}'$:

$$\begin{aligned}(V = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n), (T_i)_{i\in[\ell]}, \mathsf{ct}), ((m_i)_{i\in[\ell]}, w, r)) \in \mathcal{R}' \Leftrightarrow \\ \mathsf{ct} = \mathsf{SWE}'.\mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, (m_i)_{i\in[\ell]}); r) \text{ and } (m, w) \in \mathcal{R}, \\ \text{where } m = \sum_{i\in[\ell]} 2^{(i-1)k} m_i\end{aligned}$$

Since we only combine three NIZK proofs $(\mathsf{P}_1, \mathsf{V}_1), (\mathsf{P}_2, \mathsf{V}_2), (\mathsf{P}_3, \mathsf{V}_3)$, we can establish completeness, soundness and zero-knowledge of the resulting proof system routinely. We will argue now, that the language of $\mathsf{Prove}, \mathsf{Vrfy}$ is indeed as claimed. We consider CRS fixed. The input statement $(\mathsf{CRS}, V = (\mathsf{vk}_j)_{j\in[n]}, (T_i)_{i\in[\ell]}, \mathsf{ct})$ and witness $((m_i)_{i\in[\ell]}, w, r')$ match the structure for $\mathcal{R}'$. Let r, r_0 be as derived from r'.

P_1, V_1 establishes $\mathsf{ct} = \mathsf{SWE}'.\mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, \mathbf{m} = (m_i)_{i\in[\ell]}); r')$ for some randomness r' .

Now, P_2, V_2 establishes that there exists some $\bar{\mathbf{m}} \in \mathbb{Z}_p^\ell$ and randomnesses $r, r_0, \rho \in \mathbb{Z}_p$ such that $(c, c_0, (c'_i)_{i\in[\ell]}) = \mathsf{com}(\bar{\mathbf{m}}; r, r_0)$ and $\mathsf{C} = \mathsf{COM}(\bar{\mathbf{m}}; \rho)$. The commitment com is defined as $\mathsf{com}(\mathbf{m}; r, r_0) = (c = g_2^r, c_0 = h^r \cdot g_2^{r_0}, (c'_i = h_{T,i}^{r_0} \cdot g_T^{m_i})_i)$, where $h, (t_i)_i = H(T_i)_i^{\alpha_i}$ from the input $\mathsf{ct}, (T_i)_i$ are provided via crs and we set $(h_{T_i})_i = e(t_i, g_2)_i$. That implies by definition of $\mathsf{SWE}'.\mathsf{Dec}$, that there is some $(\bar{c_j})_{j\in[n]}, \bar{r}, \bar{a}_{i i\in[\ell]}$ such that $(h, c, c_0, (\bar{c_j})_{j\in[n]}, (c'_i, \bar{a}_i, t_i)_{i\in[\ell]}) = \mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, \bar{\mathbf{m}}); \bar{r})$.

So we receive $\mathsf{ct} = \mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, \mathbf{m} = (m_i)_{i\in[\ell]}); r')$ and $(h, c, c_0, (\bar{c_j})_{j\in[n]}, (c'_i, \bar{a}_i, t_i)_{i\in[\ell]}) = \mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, \bar{\mathbf{m}}); \bar{r})$ and $\mathsf{C} = \mathsf{COM}(\bar{\mathbf{m}}; \rho)$. Due to the perfect binding of com as discussed above, it must hold that $\bar{\mathbf{m}} = \mathbf{m}$ given that $(h, c, c_0, (c'_i, t_i)_i)$ are fixed. So, We can combine P_1, V_1 and P_2, V_2 to show there is some r', ρ such that $\mathsf{ct} = \mathsf{SWE}'.\mathsf{Enc}(1^\lambda, V, (T_i)_{i\in[\ell]}, \mathbf{m} = (m_i)_{i\in[\ell]}); r')$ and $\mathsf{C} = \mathsf{COM}(\mathbf{m}; \rho)$.

Now, P_3, V_3 ensures that there is $\mathbf{m}', \rho$ such that $\mathsf{C} = \mathsf{COM}(\mathbf{m}'; \rho)$ and $(\sum_{i\in[\ell]} 2^{(i-1)k} m'_i, w) \in \mathcal{R}$. Since C is a perfectly binding commitment, we again have $\mathbf{m}' = \mathbf{m}$. The conjunction language of $(V_1, P_1), (V_2, P_2), (V_3, P_3)$ is thus indeed identical to the one induced by $\mathcal{R}'$. This concludes our proof. $\square$

8 Implementation and Evaluation

To show the practicality of our scheme we created a prototype implementation of the base scheme SWE and evaluated its performance.

Setup In our prototype we use the noble-bls javascript library [35] that implements bilinear group operations. This library is a fast implementation of the BLS12-381 curve used in many popular cryptocurrencies like Zcash and Ethereum 2.0 for example. In the noble-bls implementation of BLS signatures the verification key is given in the group $\mathbb{G}_1$ and the signature in group $\mathbb{G}_2$. For our prototype, we used the same setup and adapted our scheme accordingly, since the groups are reversed there.

To evaluate the efficiency of our prototype we executed the script on a standard Macbook Pro with an Intel i7 processor @2,3 GHz, 16 GB of RAM using npm version 8.1.4. For benchmarking we used the micro-bmark library [34] that is also used by noble-bls.

Operation	**exp$\mathbb{G}_1$**	**exp$\mathbb{G}_2$**	**pairing**	**exp$\mathbb{G}_T$**
Time [ms]	8	33	24	21

Table 1: noble-bls execution time on the test machine

Computing Discrete Logarithms Our SWE scheme from Section 3 can be used to batch encrypt small exponents from the set $\mathbb{Z}_p$. The caveat is that at the end of decryption we do not get an element in $\mathbb{Z}_p$ but an element z_i in $\mathbb{G}_T$. To compute the actual message we have to compute the discrete logarithm of z_i to the base of g_T.

The most efficient way to compute the discrete logarithm in such a case is to use the baby-step giant-step algorithm. The key component for this algorithm is a precomputed data structure containing $(g_T^1, 1), ..., (g_T^{2^i}, 2^i)$ for some i. It is also important that this structure allows for $O(1)$ access to its elements. To this end, we used a HashMap to store the precomputed values. The map is storing data in an array of a predetermined size of $2^{32} - 1$ and uses a simple hash function to map keys (i.e. the powers of g_T) to indices of the array.

For our setup, we were able to adapt the hash function in a way that even for $i = 16$ there are no collisions in the HashMap and the discrete logarithm can be computed correctly. Below we show details about the efficiency of the precomputation step and the computation of the discrete logarithm.

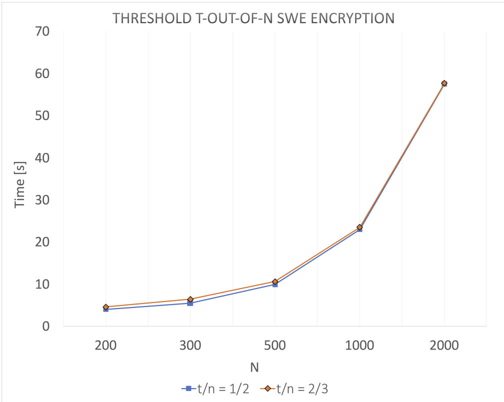
In Table 2 we show the execution time for the precomputation step and the actual baby-step giant-step computation. We benchmark two scenarios. One is a standard approach to the algorithm to divide precomputation and computation equally. In the other, we precompute more to get better efficiency of the actual computation. Based on the results we decided to encrypt messages and precompute values for $i = 16$. The experiments below use this setting. The time it takes for the baby-step giant-step algorithm to solve the DLP for a 24 bit exponent is then only 27 ms which is comparable to one pairing operation (see Table 1).

Exponent Bit Size i	**8**	**12**	**16**	**20**	**24**	**28**	**32**
Precomption	1	6	26	105	412	1622	6350
DL Computation	3	8	27	106	421	1574	5846

Table 2: Execution time in milliseconds for baby-step giant-step precomputation and discrete logarithm computation. Symmetric case with $2^{i/2}$ of precomputed exponents and $2^{i/2}$ steps of computation.

Experiments and Results We will now present evaluation results of our prototype implementation of the T-out-of-N SWE scheme. We prepared a benchmark that measures the execution time of the 4 main procedures of our SWE implementation: encryption, decryption, proof of consistency generation, and verification. For each case we compare the execution time for different thresholds: majority ($T/N = 1/2$) and supermajority ($T/N = 2/3$).

In Figure 1 we show the results for the encryption and decryption procedures. Encryption time is only slightly influenced by the different threshold parameters and takes around 1 min



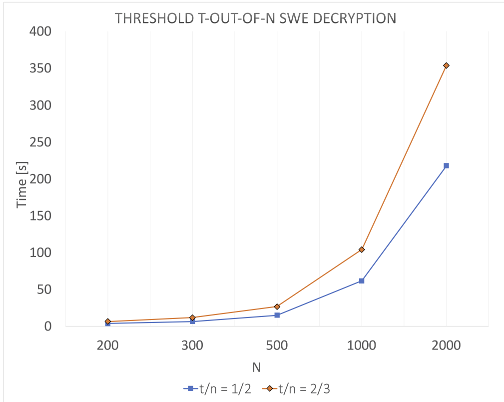
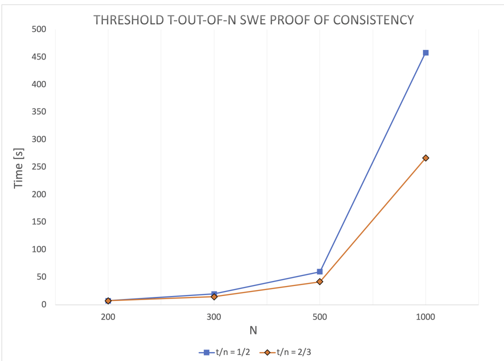


Figure 1: Average timing over 100 executions of our prototype implementation of T-out-of-N SWE encryption and decryption procedures. The X-axis is the number N of verification keys used to encrypt a 381-bit plaintext divided into 16 of 24-bit values. Data sets correspond to fractions $\frac{T}{N} = \frac{1}{2}$ and $\frac{T}{N} = \frac{2}{3}$, representing majority and supermajority.



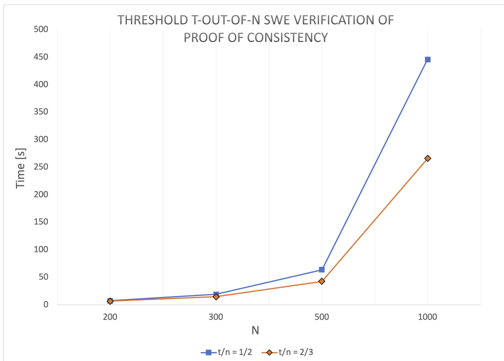


Figure 2: Average timing over 100 executions of our prototype implementation of T-out-of-N SWE procedures for creation and verification of the proof of consistency. The X-axis is the number N of verification keys used to encrypt a 381-bit plaintext divided into 16 of 24-bit values. Data sets correspond to fractions $\frac{T}{N} = \frac{1}{2}$ and $\frac{T}{N} = \frac{2}{3}$, representing majority and supermajority.

Exponent Bit Size	6	9	12	15	18	21	24
Precomption	1	5	25	99	397	1607	6338
DL Computation	2	2	3	5	8	14	27

Table 3: Execution time in milliseconds for baby-step giant-step precomputation and discrete logarithm computation. Asymmetric case with $2^{2/3i}$ of precomputed exponents and $2^{1/3i}$ steps of computation.

when encrypting for $N = 2000$ verification keys. Lowering the number N to 500 improves the execution time to around 10 second. A major difference, as to be expected, is visible in the decryption time. For a smaller threshold, T decryption is more efficient. For the case of a supermajority of $N = 2000$ decryption of a 381-bit plaintext divided into 16 blocks of 24-bit values takes around 6 min. By lowering N to 500 we can get the decryption time down to around 30 seconds.

In Figure 2 we also show our results for the procedures used to generate and verify the proof of consistency. Contrary to the decryption procedure the efficiency of generating and verifying the proof of consistency decreases with smaller T. This follows from the structure of the parity matrix used in the proof for which size increases if T decreases. Creating a proof of consistency for $N = 1000$ takes around 7 min for the majority case and around 4 min for the supermajority case. For verification of the proof, the parity matrix also has to be computed which leads to a similar execution time. Lowering N to 500 increases the efficiency to around 1 min.

9 Conclusion

We propose the McFly protocol that allows users to encrypt messages to the future. McFly is composed mainly of two components: (1) A signature-based witness encryption (SWE) - a cryptographic primitive that we introduce, and that allows messages to be encrypted with respect to a set of verification keys and a reference message; the decryption becomes possible once a threshold of the corresponding signing keys produce a signature on the reference message. (2) A BFT blockchain or a blockchain coupled with a finality layer such as Casper [15] or Afgjort [22]. By integrating the two together, with minor modifications the decryption of the SWE becomes available automatically once the blockchain committee performs its tasks; since these tasks usually happen within a predictable timeframe (e.g., block production rate), messages encrypted with the SWE scheme will only be decrypted once a predictable time has elapsed.

References

[1] Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 443–458. IEEE Computer Society Press (May 2014). DOI: 10.1109/SP.2014.35

[2] Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: TARDIS: Time and relative delays in simulation. Cryptology ePrint Archive, Report 2020/537 (2020), https://eprint.iacr.org/2020/537

[3] Bellare, M., Neven, G.: New multi-signature schemes and a general forking lemma (2017)

[4] Bellare, M., Palacio, A.: The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 273–289. Springer, Heidelberg (Aug 2004). DOI: 10.1007/978-3-540-28628-8_17

[5] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993). DOI: 10.1145/168588.168596

[6] Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., Reyzin, L.: Can a public blockchain keep a secret? In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part I. LNCS, vol. 12550, pp. 260–290. Springer, Heidelberg (Nov 2020). DOI: 10.1007/978-3-030-64375-1_10

[7] Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 757–788. Springer, Heidelberg (Aug 2018). DOI: 10.1007/978-3-319-96884-1_25

[8] Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018. pp. 435–464. Springer International Publishing, Cham (2018)

[9] Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (Aug 2001). DOI: 10.1007/3-540-44647-8_13

[10] Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (May 2003). DOI: 10.1007/3-540-39200-9_26

[11] Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (Dec 2001). DOI: 10.1007/3-540-45682-1_30

[12] Boneh, D., Naor, M.: Timed commitments. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 236–254. Springer, Heidelberg (Aug 2000). DOI: 10.1007/3-540-44598-6_15

[13] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). DOI: 10.1109/SP.2018.00020

[14] Buterin, V.: Hf1 proposal, https://notes.ethereum.org/@vbuterin/HF1_proposal

[15] Buterin, V., Griffith, V.: Casper the friendly finality gadget (2019)

[16] Camenisch, J., Kiayias, A., Yung, M.: On the portability of generalized Schnorr proofs. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 425–442. Springer, Heidelberg (Apr 2009). DOI: 10.1007/978-3-642-01001-9_25

[17] Campanelli, M., David, B., Khoshakhlagh, H., Konring, A., Nielsen, J.B.: Encryption to the future: A paradigm for sending secret messages to future (anonymous) committees. Cryptology ePrint Archive, Report 2021/1423 (2021), https://ia.cr/2021/1423

[18] Cathalo, J., Libert, B., Quisquater, J.J.: Efficient and non-interactive timed-release encryption. In: Qing, S., Mao, W., López, J., Wang, G. (eds.) ICICS 05. LNCS, vol. 3783, pp. 291–303. Springer, Heidelberg (Dec 2005)

[19] Cerulli, A., Connolly, A., Neven, G., Preiss, F.S., Shoup, V.: vetkeys: How a blockchain can keep many secrets. Cryptology ePrint Archive, Paper 2023/616 (2023), https://eprint.iacr.org/2023/616, https://eprint.iacr.org/2023/616

[20] Cheon, J.H., Hopper, N., Kim, Y., Osipkov, I.: Provably secure timed-release public key encryption **11**(2) (2008). DOI: 10.1145/1330332.1330336, https://doi.org/10.1145/1330332.1330336

[21] Deuber, D., Döttling, N., Magri, B., Malavolta, G., Thyagarajan, S.A.K.: Minting mechanism for proof of stake blockchains. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 20, Part I. LNCS, vol. 12146, pp. 315–334. Springer, Heidelberg (Oct 2020). DOI: 10.1007/978-3-030-57808-4_16

[22] Dinsdale-Young, T., Magri, B., Matt, C., Nielsen, J.B., Tschudi, D.: Afgjort: A partially synchronous finality layer for blockchains. Cryptology ePrint Archive, Report 2019/504 (2019), https://ia.cr/2019/504

[23] Dinsdale-Young, T., Magri, B., Matt, C., Nielsen, J.B., Tschudi, D.: Afgjort: A partially synchronous finality layer for blockchains. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20. LNCS, vol. 12238, pp. 24–44. Springer, Heidelberg (Sep 2020). DOI: 10.1007/978-3-030-57990-6_2

[24] ethereum.org: Ethereum 2.0 keys (2022), https://kb.beaconcha.in/ethereum-2-keys

[25] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). DOI: 10.1007/3-540-47721-7_12

[26] Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 152–168. Springer, Heidelberg (Aug 2005). DOI: 10.1007/11535218_10

[27] Gailly, N., Melissaris, K., Romailler, Y.: tlock: Practical timelock encryption from threshold bls. Cryptology ePrint Archive, Paper 2023/189 (2023), https://eprint.iacr.org/2023/189, https://eprint.iacr.org/2023/189

[28] Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 467–476. ACM Press (Jun 2013). DOI: 10.1145/2488608.2488667

[29] Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: You only speak once - secure MPC with stateless ephemeral roles. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 64–93. Springer, Heidelberg, Virtual Event (Aug 2021). DOI: 10.1007/978-3-030-84245-1_3

[30] Altair the beacon chain, https://github.com/ethereum/consensus-specs/blob/dev/specs/altair/beacon-chain.md

[31] Joux, A.: A one round protocol for tripartite diffie–hellman. In: International algorithmic number theory symposium. pp. 385–393. Springer (2000)

[32] Lenstra, A.K., Wesolowski, B.: A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366 (2015), https://eprint.iacr.org/2015/366

[33] Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Des. Codes Cryptogr. **86**(11), 2549–2586 (2018). DOI: 10.1007/s10623-018-0461-x, https://doi.org/10.1007/s10623-018-0461-x

[34] Miller, P.: micro-bmark (2022), https://github.com/paulmillr/micro-bmark

[35] Miller, P.: noble-bls12-381 (2022), https://github.com/paulmillr/noble-bls12-381

[36] Ristenpart, T., Yilek, S.: The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 228–245. Springer, Heidelberg (May 2007). DOI: 10.1007/978-3-540-72540-4_13

[37] Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., Massachusetts Institute of Technology, USA (1996)

[38] Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery **22**(11), 612–613 (Nov 1979)

[39] Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO'84. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (Aug 1984)

[40] Shanks, D.: Class number, a theory of factorization, and genera. In: Proc. of Symp. Math. Soc., 1971. vol. 20, pp. 41–440 (1971)

A Discussion on Proofs of Possession

As mentioned, the schemes extends to the case, where we only use Proofs of Possession, not Proofs of Knowledge in the underlying signature scheme Sig'. In this case, we have the following proof system:

$\mathsf{Sig}'.\mathsf{Prove}(\mathsf{vk}, \mathsf{sk})$:

- Outputs $\mathsf{Sig}'.\mathsf{Sign}^{H_{pr}}(\mathsf{sk}, \langle \mathsf{vk} \rangle)$.

$\mathsf{Sig}'.\mathsf{Valid}(\mathsf{vk}, \pi)$:

- Check whether $\mathsf{Sig}'.\mathsf{Vrfy}^{H_{pr}}(\mathsf{vk}, \langle \mathsf{vk} \rangle, \pi)$.
- If so, output 1, else $\perp$.

For our proofs based on Sig' with proofs of possession, we additionally need a version of the knowledge of exponent assumption [4] adapted to the bilinear setting. Similar to the original one this assumption holds generically:

Definition 25 (Knowledge of Exponent Assumption)**.** For a pair of groups (G_1, G_2, G_T) with generators (g_1, g_2, g_T) of prime orders p as above this assumption states that if there exists a PPT adversary $\mathcal{A}$ where:

- $\mathcal{A}$ takes as input a generator $h \in G_1$.
- $\mathcal{A}$ outputs two group elements $C \in G_1, Y \in G_2$ such that $e(h, Y) = e(C, g_2)$, that is (g_2, Y) and (h, C) have the same dlog relationship.

If such an $\mathcal{A}$ exists, then there exists a PPT extractor $\bar{\mathcal{A}}$, that takes the same input (and potentially randomness) as $\mathcal{A}$ and outputs the same C, Y and additionally c, such that $C = h^c, Y = g_2^c$.

We achieve somewhat weaker guarantees compared to Definition 8. The zero-knowledge and extractability properties additionally need an input set $\mathcal{W}$, such that for $\mathsf{vk} \in \mathcal{W}$, simulation works, while for all other keys extractability works. Simulation additionally needs an advice string, while we only need to program the oracle once. Recall $\mathcal{K}$ is the relation on public-secret key pairs. We will now state what guarantees we get for our modified BLS signature with proofs of possession:

A signature $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Agg}, \mathsf{AggVrfy}, \mathsf{Prove}, \mathsf{Valid})$ has an extractable proof of possession, if there exists a PPT simulator $S = (S_0, S_1)$ and a PPT extractor $\mathcal{E}$, such that for every at most polynomially big set $\mathcal{W}$ there is some polynomial advice-string advice such that the following holds:

- Completeness as in Definition 13.
- Zero-Knowledge: For all distinguishers $\mathcal{D}$ the following distributions are computationally indistinguishable:
  - Let H_{pr} be a random oracle. Give $\mathcal{D}$ oracle access to H_{pr} and $\mathsf{Prove}'(\cdot,\cdot)$, which responds like $\mathsf{Prove}(\mathsf{vk}, \mathsf{sk})$ on input $(\mathsf{vk}, \mathsf{sk}) \in \mathcal{K}$ for $\mathsf{vk} \in \mathcal{W}$ and with $\perp$ otherwise. Let $\mathcal{D}$ output a bit b.
  - Let $(H_0, \tau) \leftarrow S_0(\mathcal{W})$. Give $\mathcal{D}$ oracle access to H_0 and $S'(\cdot,\cdot)$, which responds like $S_1(\tau, \mathsf{vk}, \mathsf{advice})$ on input $(\mathsf{vk}, \mathsf{sk}) \in \mathcal{K}$ for $\mathsf{vk} \in \mathcal{W}$ and with $\perp$ otherwise. Let $\mathcal{D}$ output a bit b.
- Extractability: Let $(H_0, \tau) \leftarrow S_0(\mathcal{W})$. For every algorithm $\mathcal{A}$ it holds:
  Let $(H_0, \tau) \leftarrow S_0(\mathcal{W})$. Give $\mathcal{A}$ oracle access to H_0, $S'(\cdot,\cdot)$. Let $\mathcal{A}$ finally output (vk, π), and let $Q_{\mathcal{A}}$ be the queries that $\mathcal{A}$ made to H_0, $\mathsf{sk} \leftarrow \mathcal{E}(\mathsf{vk}, \pi, \tau, Q_{\mathcal{A}})$. Then

$$Pr\left[\mathsf{vk} \notin \mathcal{W} \wedge (\mathsf{vk}, \mathsf{sk}) \notin \mathcal{K} \wedge \mathsf{Valid}^{H_0}(\mathsf{vk}, \pi) = \mathsf{vk}\right] \leq negl(\lambda)$$

Theorem 19. *Given the knowledge of exponent assumption* Sig' *with* $(\mathsf{Prove}, \mathsf{Valid})$ *instantiated by the proof of possession fulfills these requirements.*

Proof. Completeness holds by the correctness of Sig'.

Now let us move on: Let $\mathcal{W}$ and λ be given. We assume the experiment receives/knows the generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$. We take an optional second group element $h \in \mathbb{G}_1$ as parameter or choose h randomly ourselves. We define the simulator S_0: It takes a pseudorandom function family PRF_k with $k \in \{0,1\}^a$ such that the domain is large enough for any input $\langle \mathsf{vk} \rangle$ for vk generated by $\mathsf{KeyGen}(1^\lambda)$ and output mapped into $\mathbb{Z}_p$. It chooses $k \leftarrow_\$ \{0,1\}^a$, sets

$$H_0(m) = \begin{cases} g_1^{\mathsf{PRF}_k(m)} & \text{if } m \text{ represents } \langle \mathsf{vk} \rangle \text{ for } \mathsf{vk} \in \mathcal{W} \\ h^{\mathsf{PRF}_k(m)} & \text{otherwise} \end{cases}$$

and outputs $\tau = k$ as information to S_1. Let us now argue zero-knowledge holds: We construct S_1 as follows: $S_1(k, \mathsf{vk}, \mathsf{advice})$ retrieves $r = \mathsf{PRF}_k(\langle \mathsf{vk} \rangle)$ and outputs $(H_0, a(\mathsf{vk})^r)$ for advice being a function $a : \mathcal{W} \to \mathbb{G}_1$, such that for every $\mathsf{vk} = g_2^{\mathsf{sk}} \in \mathcal{W}$, $a(\mathsf{vk}) = g_1^{\mathsf{sk}}$.

In a first hybrid, we can replace H_{pr} by H_0 in the first distribution. $\mathcal{D}$ can not detect the change except with negligible probability, or else it could break pseudorandomness of PRF. Since we only have to regard $\mathsf{vk} \in \mathcal{W}$, as other inputs prompt the return of $\perp$ in both distributions, it holds $a(\mathsf{vk})^r = g_1^{\mathsf{sk} \cdot r} = H_0(\langle \mathsf{vk} \rangle)^{\mathsf{sk}}$. The distributions are then identical.

It remains to show extractability. Let us assume there is an algorithm $\mathcal{A}$ that produces an accepting output (vk, π) with $\mathsf{vk} \notin \mathcal{W}$ with non-negligible probability. We argue that we can build an adversary that internally runs $\mathcal{A}$.

The adversary $\mathcal{A}'$ receives a generator $h' \in \mathbb{G}_1$. It runs the experiment with $h = h'$ as the optional input. It runs $S_0(\mathcal{W})$ and retrieves H_0, k.

It interacts with $\mathcal{A}$ like the experiment would, getting the queries $Q_{\mathcal{A}}$ made by $\mathcal{A}$ and the outputs vk, π. The simulation via S works without any issues as simulateable and extractable keys are in distinct domains.

If $\mathcal{A}$ produces an accepting output vk, π for $\mathsf{vk} \notin \mathcal{W}$, it must hold $e(\pi, g_2) = e(H_0(\langle \mathsf{vk} \rangle), \mathsf{vk}) = e(h^{\mathsf{PRF}_k(m)}, \mathsf{vk})$. Thus, (g_2, vk) and $(h^{\mathsf{PRF}_k(m)}, \pi)$ share the same dlog relationship.

As we can see, this constitutes an adversary in the knowledge of exponent assumption. Therefore, we are guaranteed the existence of an efficient extractor $\bar{\mathcal{A}}'$ that comes to the same output as $\mathcal{A}'$ - namely $\pi \in \mathbb{G}_1, \mathsf{vk} \in \mathbb{G}_2$ and additionally outputs x such that $\mathsf{vk} = g_2^x$.

$\square$

Now if we want to use this in our proofs, instead of the Schnorr based variant, we essentially set $\mathcal{W}$ as the keys made by the reduction and then can extract for all keys chosen by the adversary. There are some subtleties to this however; in the proof of unforgeability above, for example, we may sometimes have to give a proof for vk^* and sometimes the adversary may use vk^* itself. To deal with this, we guess in the beginning, which case happens and set $\mathcal{W} = \emptyset$ or $\mathcal{W} = \{\mathsf{vk}^*\}$ with probability $1/2$, introducing a factor $1/2$ to our success probability. We constructed all our proofs such that the advice string can always be constructed, as for all $\mathsf{vk} = g_2^{\mathsf{sk}}$ we give out in reductions, we make sure they already know g_1^{sk}.