McFly: Verifiable Encryption to the Future Made Practical

Nico Döttling[1](✉), Lucjan Hanzlik[1], Bernardo Magri[2], and Stella Wohnig[1,3](✉)

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{doettling,hanzlik,stella.wohnig}@cispa.de

[2] The University of Manchester, Manchester, UK
bernardo.magri@manchester.ac.uk

[3] Saarland University, Saarbrücken, Germany

Abstract. Blockchain protocols have revolutionized how individuals and devices 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.

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, block-

N. Döttling—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.

chain 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 [24] without a trusted third party.

In previous works [3,15], 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

[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.

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.

1.1 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 [11] or Afgjort [16] 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. We instantiate SWE with an aggregatable multi-signature scheme that is a BLS scheme [6] 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 [19] that are highly inefficient, we demonstrate SWE to be practicable. To ensure that decryption is always possible we make SWE verifiable by designing specially tailored proofs 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

[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.

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 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 [20]). 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.

A blockchain functionality modelling the above requirements and an analysis on how to integrate our scheme with a modified version of Ethereum 2.0 can be found in the full version. Intuitively, to make Ethereum 2.0 running with Casper [11] compatible with our 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, which

[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.

were only introduced in Ethereum Altair [10], we can have a horizon of up to 27 hours. However, it is unclear whether sync committees enjoy the same level of trust as 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. Details can be found in the full version.

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.

SWE Based on BLS. Our construction of Signature-based Witness Encryption is based on the BLS signature scheme [7] and its relation to identity-based encryption [5]. 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 [5]. Specifically, in the IBE scheme of [5] 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 [6]. 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 [25] 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 [21]. 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 routinely.

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 Sect. 2.1 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 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 [27] to $O(2^{k/2})$ group operations. This leads to a very efficient implementation as a 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 in the full version.

- 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.
- 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.
- We can now rely on efficient and succinct proof systems such as Bulletproofs [9] to establish additional guarantees about the encrypted plaintext. For instance, we can rely on the range-proofs of [9] to ensure that the encrypted messages are within a certain range to ensure that our BSGS decryption procedure will recover the correct plaintext.

[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 [9,18] we refer to them as proof systems.

To make this construction efficient, we add redundancy to include 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 [24]. 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 [24] 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, which 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 [8], 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 its 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.

Our approach is closer to the paradigm of using a trusted party as in [13,14]. 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 [13] 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) servers 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.

In [22] 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, [12] 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 [2], 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 [20] 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.

BLS Signatures and Identity-Based Encryption (IBE). The BLS signature scheme, introduced in [7], 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 [6]. Due to the very space-efficient aggregation, BLS signatures are used in widely deployed systems such as Ethereum 2.0 [17]. Aggregation for potential duplicate messages is achieved in [4,23].

Identity based encryption was first introduced by Shamir [26]. 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 Contents

This is a shortened conference-version of this paper including an overview of our results, the construction of our modified BLS multi-signature scheme, as well as definitions and constructions for both our basic SWE and McFly. Due to space limitations, we refer readers to the full version for more details including:

– The construction of a proof compatibility layer to add verifiability to SWE.
– A blockchain functionality rigorously modelling the requirements on the blockchain that are outlined above.
– Details and evaluations of an implementation of our scheme.
– Discussions of applications of our scheme in decentralized auctions and randomness beacons.

1.5 Preliminaries

We denote by $\lambda \in \mathbb{N}$ the security parameter and by $x \leftarrow \mathcal{A}(\mathsf{in}; r)$ the output of the randomized algorithm $\mathcal{A}$ on input in 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. 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 $[n]$. PPT denotes probabilistic polynomial time. Also, $poly(x), negl(x)$ respectively denote any polynomial or negligible function in parameter x.

We assume familarity with the following cryptographic notions, for which full definitions are included in our full version: Aggregateable multi-signatures, Cryptographic Hash functions, Pseudo-random functions, commitment schemes, Zero-Knowledge Proofs (of Knowledge), Secret Sharings, Reed-Solomon Codes, Lagrange Interpolation, Bilinear Maps as well as the Co-Diffie-Hellman and Bilinear Diffie-Hellman Assumptions.

2 Signature-Based Witness Encryption

In this section we introduce the new cryptographic primitive SWE that is the core technical component of the McFly protocol. We formally define it next.

Definition 1 ***(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.*

We require such a scheme to fulfill robust correctness and security.

Definition 2 ***(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 3 ***(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* $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ *and provides* vk_i *as well as* $\mathsf{Sig.Prove}^{H_{pr}}(\mathsf{vk}_i, \mathsf{sk}_i)$ *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* $i \in [t-1]$*,* $\mathsf{Sig.Valid}(\mathsf{vk}_i, \pi_i) = 0$*, 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* $\mathsf{Sig.Sign}(\mathsf{sk}_i, T)$*.*
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* $\mathsf{Sig.Sign}(\mathsf{sk}_i, T)$*, else it aborts.*
8. *Finally,* $\mathcal{A}$ *outputs a guess* b'*.*
9. *If* $b = b'$*, the experiment outputs 1, else 0.*

Definition 4 (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:*

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

2.1 Construction

In the following, we describe a t-out-of-n SWE. Let two base groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order p with generators g_1, g_2 which have a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ into a target group $\mathbb{G}_T$ with generator g. Also, we assume full-domain hash functions $H : \{0,1\}^* \rightarrow \mathbb{G}_1$, $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p$ and $H_{pr} : \{0,1\}^* \rightarrow \mathbb{Z}_p$.

First, let us describe the underlying signature scheme Sig'.

Protocol Sig'

```
$\mathsf{Sig}'.\mathsf{KeyGen}(1^\lambda)$: Randomly pick $x \leftarrow_\$ \mathbb{Z}_p$ and output $(\mathsf{vk} = {g_2}^x, \mathsf{sk} = x)$.
$\mathsf{Sig}'.\mathsf{Sign}(\mathsf{sk}, T)$: Output $H(T)^{\mathsf{sk}}$.
$\mathsf{Sig}'.\mathsf{Vrfy}(\mathsf{vk}, T, \sigma)$: If $(e(\sigma, g_2) = e(H(T), \mathsf{vk}))$, output 1, else output 0.
$\mathsf{Sig}'.\mathsf{Agg}((\sigma_1, \dots, \sigma_k), (\mathsf{vk}_1, \dots, \mathsf{vk}_k))$:
    – 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 $\sigma \leftarrow \prod_{i \in [k]} \sigma_i^{L_i}$.
$\mathsf{Sig}'.\mathsf{AggVrfy}(\sigma, (\mathsf{vk}_1, \dots, \mathsf{vk}_k), (T_1, \dots, T_k))$:
    – 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.$\mathsf{Valid}^{H_{pr}}(\mathsf{vk}, \pi)$.
```

Here, Schnorr.Prove, Schnorr.Valid are the non-interactive variant of the well-known Schnorr proofs due to Fischlin [18]. As shown in [18], they constitute an online-extractable proof of knowledge for the key relation $\mathcal{K} = \{(g^x, x) : x \in \mathbb{Z}_p\}$.

Theorem 1. Sig' *is a correct aggregatable multi-signature scheme.* Sig' *is unforgeable, assuming that* H *is modelled as a random oracle and that the computational Co-Diffie-Hellman assumption holds for* $(\mathbb{G}_1, \mathbb{G}_2)$.

The proof is given in the full version only, due to space restrictions. Now, we can give the construction of our SWE scheme.[6]

Protocol SWE for signature scheme Sig'

```
$\mathsf{SWE}'.\mathsf{Enc}(1^\lambda, (\mathsf{vk}_j)_{j \in [n]}, (T_i)_{i \in [\ell]}, (m_i)_{i \in [\ell]})$:
    – 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 $U \not\subseteq V$, abort.
    – Define as $I$ the indices $j \in [n]$ s.t. $\mathsf{vk}_j \in U$.
```

[6] Notice that a previous version of this manuscript provided a slightly different protocol, which had the caveat, that all T_i needed to be distinct.

```
– 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, that we only do the expensive computation of c^* in SWE.Dec once.[7] Further, we require that all T_i are from the range $\{0, \ldots, 2^k - 1\}$ for some k to enable efficient discrete log computation via the baby-step giant-step method.

Theorem 2. *The following statements hold:*

1. SWE *for the signature scheme* Sig′ *has robust correctness, given that H_2 is collision resistant.*
2. *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.*
3. *There are protocols* SWE.Prove, SWE.Vrfy *which extend* SWE *to be verifiable.*

The proofs of statement 1, 2 are found in the full version only, due to space restrictions. For statement 3, the full version includes a full construction and proofs of SWE.Prove, SWE.Vrfy which closely follows the outline given in Sect. 1.2.

Efficiency of SWE. 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 with cheaper group operations. In Table 1, we briefly analyze the number of group operations in each group required for encryption and decryption. We regard the numbers n, ℓ to be fixed and give upper bounds on the operations needed. Note also, that the extraction of the discrete logarithm does not cause a large overhead as we use the baby-step giant-step methodology. More details and a concrete performance evaluation for an implementation of our scheme can be found in the full version.

[7] In case the sets of signers are the same for all T_i. Otherwise we compute it once per relevant set U of signers.

Table 1. Analysis of SWE Efficiency in Group Operations

		encryption	decryption
Evaluations of H, H_2		ℓ, n	$0, n$
Multiplications, Exponentiations	in $\mathbb{G}_1$	$0, \ell$	$0, 0$
	in $\mathbb{G}_2$	$n, 2 + 2n + \ell$	$n - 1, n$
	in $\mathbb{G}_T$	ℓ, ℓ	$2\ell, 0$
Pairing Evaluations		ℓ	2ℓ
dlog in $\mathbb{G}_T$		0	ℓ

3 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.

3.1 Formal Model and Guarantees

In the full version we introduce a simplified model for blockchains in the form of the $\mathcal{BC}_{\lambda,H}$ functionality reflecting the requirements introduced in Sect. 1.1. It essentially runs a blockchain with a static committee of size $n = poly(\lambda)$. The public interface allows to retrieve the committee keys and the published blocks. The adversary is allowed a (static) corruption threshold $c < n/2$. They may control c committee members and choose the block contents to be signed.

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 $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Vrfy})$ in a hybrid model where access to the public interface of $\mathcal{BC} = \mathcal{BC}_{\lambda,H}$ is assumed. 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 5 (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 6 (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' *while the number of finalized blocks* $ctr < d$ *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.*

Definition 7 (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 relation* $\mathcal{R}'$*:*

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

$\mathsf{Enc}(\ldots; r, V)$ denotes, that the randomness used is r and the committee 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.

3.2 Protocol Description

Let $\mathsf{COM} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Vrfy})$ 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. 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 commitee keys $V$ by calling $\mathsf{QueryKeys}$ to $\mathcal{BC}$.
    – Split $m = (m_i)_{i\in[\ell]}$ for $m_i \in \{0, \dots, 2^k - 1\}$ s.t. $m = \sum_{i\in[\ell]} 2^{(i-1)k} m_i$.
    – Output $\mathsf{ct} \leftarrow \mathsf{SWE.Enc}(1^\lambda, V, (H(d))_{i\in[\ell]}, (m_i)_{i\in[\ell]})$.
$\mathsf{McFly.Dec}^{\mathcal{BC}}(\mathsf{ct}, d)$:
    – If $\mathsf{QueryTime}$ returns less than $d$, abort.
    – Get $(\sigma, U)$ 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 $b \leftarrow \mathsf{SWE.Vrfy}(\mathsf{CRS}, V, (H(d))_{i\in[\ell]}, \mathsf{ct}, \pi)$
```

Theorem 3. McFly *is correct, given that* SWE *has robust correctness.* McFly *is secure given that* SWE *is secure and H is collision resistant.* McFly *is verifiable, given that* SWE *is a verifiable SWE.*

The proofs are only included in the full version due to space restrictions.

Extension for Dynamic Committees. In our model, we assumed static committees. However, finality layers advocate for a short-lived dynamic committee, as committee members usually become targets of attacks. 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.

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. https://doi.org/10.1109/SP.2014.35
2. Benhamouda, F., et al.: Can a public blockchain keep a secret? In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12550, pp. 260–290. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64375-1_10
3. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 757–788. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_25

4. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 435–464. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_15
5. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_13
6. 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 (2003). https://doi.org/10.1007/3-540-39200-9_26
7. 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 (2001). https://doi.org/10.1007/3-540-45682-1_30
8. Boneh, D., Naor, M.: Timed commitments. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 236–254. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_15
9. 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. https://doi.org/10.1109/SP.2018.00020
10. Buterin, V.: Hf1 proposal. https://notes.ethereum.org/@vbuterin/HF1_proposal
11. Buterin, V., Griffith, V.: Casper the friendly finality gadget (2019)
12. 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
13. 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 (2005)
14. Cheon, J.H., Hopper, N., Kim, Y., Osipkov, I.: Provably secure timed-release public key encryption. ACM Trans. Inf. Syst. Secur. **11**(2) (2008). https://doi.org/10.1145/1330332.1330336
15. 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 (2020). https://doi.org/10.1007/978-3-030-57808-4_16
16. 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
17. ethereum.org: Ethereum 2.0 keys (2022). https://kb.beaconcha.in/ethereum-2-keys
18. 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 (2005). https://doi.org/10.1007/11535218_10
19. 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, June 2013. https://doi.org/10.1145/2488608.2488667

20. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: you only speak once. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 64–93. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_3
21. Joux, A.: A one round protocol for tripartite Diffie–Hellman. In: Bosma, W. (ed.) ANTS 2000. LNCS, vol. 1838, pp. 385–393. Springer, Heidelberg (2000). https://doi.org/10.1007/10722028_23
22. Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Des. Codes Cryptogr. **86**(11), 2549–2586 (2018). https://doi.org/10.1007/s10623-018-0461-x
23. 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 (2007). https://doi.org/10.1007/978-3-540-72540-4_13
24. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, USA (1996)
25. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery **22**(11), 612–613 (1979)
26. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_5
27. Shanks, D.: Class number, a theory of factorization, and genera. In: Proc. of Symp. Math. Soc., 1971, vol. 20, pp. 41–440 (1971)