Forward-Secure Public Key Encryption without Key Update from Proof-of-Stake Blockchain

Seiya Nuta[1], Jacob C. N. Schuldt[2], and Takashi Nishide[3]

[1] University of Tsukuba, Japan
nuta@seiya.me
[2] National Institute of Advanced Industrial Science and Technology, Japan
jacob.schuldt@aist.go.jp
[3] University of Tsukuba, Japan
nishide@risk.tsukuba.ac.jp

Abstract. A forward-secure public-key encryption (PKE) scheme prevents eavesdroppers from decrypting past ciphertexts in order to mitigate the damage caused by a potential secret key compromise. In prior works, forward security in a non-interactive setting, such as forward-secure PKE, is achieved by constantly updating (secret) keys. In this paper, we formalize the notion of blockchain-based forward-secure PKE and show the feasibility of constructing a forward-secure PKE scheme without key update (i.e. both the public key and the secret key are immutable), assuming the existence of a proof-of-stake blockchain with the distinguishable forking property introduced by Goyal *et al.* (TCC 2017). Our construction uses the proof-of-stake blockchain as an *immutable decryption log* and witness encryption by Garg *et al.* (STOC 2013) to ensure that the same ciphertext cannot be decrypted twice, thereby rendering a compromised secret key useless with respect to decryption of past ciphertext the legitimate user has already decrypted.

Keywords: Public-Key Encryption · Forward Security · Blockchain

1 Introduction

Forward security for public-key encryption is a security notion that ensures that a secret key compromise does not affect the confidentiality of past ciphertexts. More specifically, even if Alice's long-term secret key sk_A is compromised by an eavesdropper Eve, who observed and recorded ciphertexts sent to Alice in the past, forward security guarantees that Eve does not learn the secrets required for decrypting these *past* ciphertexts (i.e. sk_A is insufficient to decrypt).

While forward security in an interactive setting (e.g. key exchange protocols), can be achieved relatively easily by generating *ephemeral* secrets that are erased when no longer needed, this is harder in a non-interactive setting. However, one

This work appears in INDOCRYPT 2021 [NSN21].
https://doi.org/10.1007/978-3-030-92518-5_20

strategy for achieving forward security in a non-interactive setting, is to constantly update or erasing long term secrets. For example, a naïve approach to obtaining forward-secure PKE is generating a series of *one-time* public/secret key pairs; once a key pair has been used, erase the secret key as soon as possible to ensure that an adversary cannot learn this key in a potential future compromise. The disadvantage of this approach is that a sender needs to update Alice's public key if all of the key pairs have been used, and furthermore needs to be aware of which keys Alice has already used and erased. This makes the naïve approach impractical, but more practical approaches to forward security have been developed, which we will briefly outline below.

Canetti *et al.* [CHK03] formally introduced forward-secure PKE by extending the definition of PKE with a key update algorithm. In their scheme, the encryption algorithm takes as input a time period along with the receiver's public key and a message. The ciphertext is associated with the specified time period. The key update algorithm takes as input a secret key sk and outputs an updated secret key sk' (the public key pk remains the same). Even if an adversary compromises sk', they cannot decrypt ciphertexts in the prior periods (and thus provides forward security).

Green *et al.* [GM15] presented a fine-grained forward-secure (aka. absolute forward security [BG20]) encryption scheme called puncturable encryption. It introduces a key update algorithm similar to [CHK03], but allows revoking a *specific* ciphertext, that is, the key update algorithm outputs an updated secret key which can be used to decrypt ciphertexts *except* the ciphertext given to the algorithm.

While interactive by definition, the recent work [GHJL17, DJSS18, AGJ19] on ensuring forward security of 0-RTT key exchange involves techniques that can be used to implement forward security for non-interactive primitives such as PKE. The idea behind 0-RTT key exchange, introduced in such as TLS 1.3 [Res], is to enable clients to send encrypted data in their first message using pre-shared secrets. This essentially corresponds to a non-interactive encryption for the server, and to provide forward security of this data, is almost equivalent to constructing a forward-secure PKE (e.g. see the bloom filter encryption in [DJSS18]).

To the best of our knowledge, forward security in a non-interactive setting such as PKE, has only been achieved by introducing *key update* [CHK03, GM15, DJSS18, AGJ19, GHJL17, PS14]. This seems natural, since in an ordinary PKE, if an eavesdropper Eve compromises the secret key sk, she can decrypt any ciphertext c by simply running the decryption algorithm Dec to obtain $m \leftarrow \mathsf{Dec}(sk, c)$. Hence, in order to achieve forward security, it is natural to prevent Eve from compromising an *unmodified* secret key. In the key update approach, we update or partially break the secret key $sk^{(i)}$ to derive a new secret key $sk^{(i+1)}$ which cannot be used for decrypting past (or already decrypted) ciphertexts, and then erase the old key $sk^{(i)}$.

1.1 Our Contribution

In this paper, we give a feasibility result of a forward-secure encryption scheme *without key update*, i.e. both the public key and the secret key remain unchanged like ordinary (non-forward-secure) PKE. To achieve this, we allow the PKE scheme to make use of a blockchain for encryption and decryption of messages.

Firstly, we note that the standard definitions of correctness and (forward) security are insufficient for capturing a setting in which the PKE scheme depends on a blockchain. This is due to the ability of an adversary to observe any information posted to the blockchain when encrypting or decrypting messages, and the ability to post maliciously crafted blocks to the blockchain, which might prevent an honest user from correctly decrypting a ciphertext. Hence, we appropriately extend these definitions. Our forward security notions are obtained by extending the standard IND-CPA security notion for ordinary PKE with two additional oracles. The first oracle, Leak, captures secret key leakage that happens after the honest user decrypts the challenge ciphertext. The second oracle, $\mathsf{HonestDec}$, captures the information leakage an adversary can observe on the blockchain when an honest user decrypts a ciphertext. For the full details of our security definition, see Section 4.

Our construction of a forward-secure PKE without key update, assumes the existence of a proof-of-stake blockchain which satisfies properties described in [GG17]. We combine this with witness encryption, which in general allows a plaintext to be encrypted under an NP statement instead of a public encryption key, and anyone in possession of a witness for the statement, will be able to decrypt. In our construction, we use witness encryption to tie a ciphertext to information posted by the decryptor to the blockchain, and thereby turn the blockchain into an *immutable decryption log* that only allows a ciphertext to be decrypted once. In other words, like puncturable encryption [GM15], our construction implements fine-grained forward security which removes the ability to decrypt on a ciphertext-by-ciphertext basis, as opposed to the (standard) more coarse approach of revoking the ability to decrypt any ciphertext constructed in the time period between key updates (we discuss security further in Section 4). Note that while encryptor and decryptor are required to interact with the blockchain protocol to obtain an updated view of the blockchain, the communication between the two remains non-interactive: once the encryptor has created a ciphertext based on his current view of the blockchain, no further communication is required on his part and he can go offline without affecting the decryptor's ability to decrypt.

Specifically, the pair of public key and secret key in our construction is simply that of a digital signature scheme. The encryption algorithm uses witness encryption to encrypt a message for an NP statement capturing that a certain type of message signed by the receiver has been posted to the blockchain. The decryption algorithm generates an ephemeral secret esk, posts a signed message associated with esk to the blockchain, which will allow the decryption of the ciphertext using the relevant sequence of blocks on the blockchain and esk as a witness (decryption key). Immediately after decryptions, the decryption algo-

rithm erases esk which ensures that an adversary compromising the secret key, will not be able to decrypt as they don't know esk.

Since our construction uses a simple key pair of a digital signature scheme and these are immutable, the size of the keys is obviously independent of the number of time periods or decryptions unlike existing forward-secure PKE schemes [CHK03, GM15]. Fixed immutable keys are furthermore an interesting property from an application point of view. For example, it is undesirable to use a fine-grained forward-secure PKE scheme with key updates in a scenario where the decryption key is used by multiple devices, such as laptops and smartphones, as keys would have to be synchronized to maintain fine-grained forward security. This concern is alleviated by fixed immutable secret keys. Lastly, we note that our construction enjoys some interesting security properties in addition to forward security, such as secret key leakage detection and a variant of post-compromise security [CCG16]. We will discuss these benefits in detail in Section 5.

2 Preliminaries

In this section, we introduce building blocks and their security definitions. Besides the primitives defined below, we make use of an EUF-CMA secure signature scheme $\mathsf{Sig} = (\mathsf{Sig.KGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Ver})$, and a one-way hash function $H : \{0,1\}^n \to \{0,1\}^m$. Due to space limitations, we do not include the standard definitions of these, but defer these to the full version of the paper.

2.1 Witness Encryption

Witness Encryption is a type of encryption introduced by Garg *et al.* [GGSW13]. Instead of a pair of public and private keys, in witness encryption, a plaintext is encrypted with respect to an NP statement x and the ciphertext can be decrypted with the corresponding witness w.

Definition 1 (Witness Encryption [GGSW13]). *A witness encryption scheme* WE *for NP language* L *(with witness relation* R*) is a tuple of algorithms* $(\mathsf{WE.Enc}, \mathsf{WE.Dec})$.

- $c \leftarrow \mathsf{WE.Enc}(1^\lambda, x, m)$*: The encryption algorithm* $\mathsf{WE.Enc}$ *takes as input a string* x*, and a message* m*, and outputs a ciphertext* c*.*
- $m/\perp \leftarrow \mathsf{WE.Dec}(c, w)$*: The decryption algorithm* $\mathsf{WE.Dec}$ *takes as input a ciphertext* c*, and a string* w*, and outputs a message* m *or the symbol* $\perp$*.*

A witness encryption scheme WE *is required to satisfy correctness: for all security parameters* λ*, all strings* x *and* w *for which* $R(x, w)$ *holds, for all* m*, it holds that* $\mathsf{WE.Dec}(\mathsf{WE.Enc}(1^\lambda, x, m), w) = m$.

For a witness encryption scheme, we will use the security notion *extractability*, first proposed in [GKP+13], which informally requires that, for all adversaries able to distinguish between encryptions of different messages for a statement x,

there exists an extractor that can extract a witness w from the adversary, such that $R(x, w)$ holds. We use the adaptive definition by Bellare *et al.* [BH15] in which $\mathcal{A}$ is allowed to specify x.

Definition 2 (Extractability). *A witness encryption scheme with witness relation R is extractable if for every security parameter λ, every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with a random tape r, there exists a corresponding PPT algorithm $\mathcal{E}$ (the extractor) such that:*

$$\left| \Pr\left[b' = b \wedge \neg R(x,w) \;\middle|\; \begin{array}{l} (x, m_0, m_1, st) \leftarrow \mathcal{A}_1(1^\lambda; r); \\ b \xleftarrow{\$} \{0,1\}; \\ c \leftarrow \mathsf{WE.Enc}(1^\lambda, x, m_b); \\ b' \leftarrow \mathcal{A}_2(st, c); \\ w \leftarrow \mathcal{E}(1^\lambda, r); \end{array} \right] - \frac{1}{2} \right| \leq neg(\lambda)$$

The above definition ensures that if an adversary $\mathcal{A}$ with non-negligible advantage $\varepsilon(\lambda)$ in distinguishing the ciphertexts of two messages exists, an extractor $\mathcal{E}$ with success probability $\varepsilon(\lambda) - neg(\lambda)$ must also exist.

Instantiating Witness Encryption. Witness encryption is a strong cryptographic primitive and efficiently instantiating this remains a work in progress. Recent interesting results include constructions by Barta *et al.* [BIOW20] based on the generic group model, and Bartusek *et al.* [BIJ+20] based on affine determinant programs, with the latter claimed to be the first construction sufficiently efficient to be implementable. However, these works do not consider extractability, and it is unclear whether efficient extractors can be obtained for these construction.

Goldwasser *et al.* [GKP+13] proposed a candidate extractable witness encryption scheme but without a formal security reduction. Liu *et al.* [LJKW18] proposed a construction based on multi-linear maps, which can be instantiated from indistinguishability obfuscation (iO) [AFH+20], which in turn can be obtained from well-founded assumptions [JLS21], leading to a theoretical instantiation. A different approach was taken by Goyal *et al.* [GKM+20] who show how the functionality of extractable witness encryption can be implemented efficiently on a blockchain. This approach is especially appealing in relation to our work due to the obtained efficiency and that our construction already makes use of a blockchain. Note that [GKM+20] requires the miners maintaining the blockchain to implement additional functionality i.e. smaller changes would have to be made to existing blockchain protocols to achieve the desired functionality, and to maintain forward security, the communication between miners and the decryptor must be forward secure (e.g. by using TLS 1.3 [Res]). Furthermore, due to the dependency on a blockchain, [GKM+20] does not follow the standard definition of witness encryption. However, in this work, we will make use of the standard definitions above.

2.2 Blockchain Protocol

In general, a blockchain protocol is a multi-party distributed protocol that maintains an ordered sequence of blocks (*blockchain*) without a trusted third party. The blockchain is continuously extended by parties called *miners* under a consensus algorithm and forging sufficiently old blocks is considered difficult based on underlying hardness assumptions. A *Proof-of-Stake* blockchain uses a consensus algorithm in which a party with more stake (e.g. number of coins) is more likely to succeed in mining a new block. Below, we recall the abstract definition of blockchain protocols used in [GG17].

Definition 3 (Blockchain Protocol). *A blockchain protocol* BLC_V *with validity predicate* V *is a tuple of algorithms* $(\mathsf{BLC}_V.\mathsf{UpdateState}, \mathsf{BLC}_V.\mathsf{GetRecords}, \mathsf{BLC}_V.\mathsf{Broadcast})$.

- $\mathsf{BLC}_V.\mathsf{UpdateState}(1^\lambda)$*: It is a stateful algorithm that takes as input the security parameter* λ *and maintains the local state* st*. It has no output.*
- $\mathbf{B} \leftarrow \mathsf{BLC}_V.\mathsf{GetRecords}(1^\lambda, \mathsf{st})$*: It takes as input the security parameter* λ *and a local state* st*, and outputs the longest ordered sequence of blocks* $\mathbf{B}$ *(the* blockchain*) contained in* st*.*
- $\mathsf{BLC}_V.\mathsf{Broadcast}(1^\lambda, m)$*: It takes as input the security parameter* λ *and a message* m*, and spreads the message* m *over the blockchain network. It outputs nothing.*

In the above, V is a predicate which takes a sequence of blocks $\mathbf{B}$ and outputs 1 if $\mathbf{B}$ is valid. The definition of "validity" varies with the blockchain protocol; details of how V is defined will not be important for our purpose.

Blockchain Execution. At a high level, the execution of the blockchain protocol corresponds to the participants running $\mathsf{UpdateState}$, which will continuously update their state according to messages broadcast using $\mathsf{Broadcast}$ e.g. a miner might broadcast a new successfully mined block. Each participant can access his current view of the blockchain via $\mathsf{GetRecords}$. We assume that (honest) miners will include any record broadcast via $\mathsf{Broadcast}$ in the blocks they attempt to mine, which allows all participating parties to have records added to the blockchain (e.g. in cryptocurrencies, a user might wish to add a transaction).

In [GG17], the execution of a blockchain protocol is formally modeled in the UC-framework [Can01], and is directed by the environment $\mathcal{Z}$, which initially activates all participants as either honest or corrupt (as in [GG17], we only consider static corruptions). All corrupt parties are controlled by an adversary $\mathcal{A}$. The execution starts by all honest users running $\mathsf{UpdateState}$ on an empty state, and proceeds in rounds. In each round, an honest user might receive a record from $\mathcal{Z}$ which it will attempt to add to the blockchain, as well as messages from the other parties. The user may then perform any computation, broadcast a message via $\mathsf{Broadcast}$, and update its local state. $\mathcal{A}$ is responsible for delivering all messages between parties, and may delay or reorder these, but is not allowed

to modify them. $\mathcal{Z}$ can communicate with $\mathcal{A}$ and access the local view of the blockchain obtained via $\mathsf{GetRecords}$ of any honest party. For a more detailed discussion of the blockchain execution, see [GG17].

We will let $\mathsf{EXEC}^{\mathsf{BLC}_V}[\mathcal{A}, \mathcal{Z}, 1^\lambda]$ denote the above execution, and $\mathsf{view} \leftarrow \mathsf{EXEC}^{\mathsf{BLC}_V}[\mathcal{A}, \mathcal{Z}, 1^\lambda]$ denote the joint view of all parties in the execution. The latter fully determines the former.

Blockchain Properties. We will now define several blockchain properties introduced in [GG17], which our construction will be based on. In these definition, we make use of the *unique stake fraction* of the last ℓ blocks of a blockchain $\mathbf{B}$, which we denote $\mathsf{u\text{-}stakefrac}(\mathbf{B}, \ell)$, and which is defined to be the combined stake of all miners who mined at least one of the last ℓ blocks in $\mathbf{B}$ divided by the total amount of stake for the blockchain. Additionally, we will use the notation $\mathbf{B}^{\lceil \ell}$ to denote $\mathbf{B}$ with the last ℓ blocks removed, and $\mathbf{B} \preceq \tilde{\mathbf{B}}$ to denote that $\mathbf{B}$ is a prefix of $\tilde{\mathbf{B}}$.

The blockchain properties are defined based on the following predicates: blockchain consistency ($\mathsf{consistent}$), which captures that all honest participants in the blockchain protocol agrees upon all except the last ℓ blocks; sufficient stake contribution ($\mathsf{suf\text{-}stake}$), which captures that all blockchains of length ℓ has a unique stake fraction of at least β; and bounded stake forking ($\mathsf{bd\text{-}stake\text{-}fork}$), which captures that all maliciously constructed forks of the blockchain has unique stake fraction less than α. Formally, these predicates are defined as:

- $\mathsf{consistent}^\ell(\mathsf{view}) = 1$ iff for all rounds $r \leq \tilde{r}$ and honest parties i, j in view with blockchain $\mathbf{B}$ in round r and $\tilde{\mathbf{B}}$ in round $\tilde{r}$, respectively, it holds that $\mathbf{B}^{\lceil \ell} \preceq \tilde{\mathbf{B}}$.
- $\mathsf{suf\text{-}stake}^\ell(\mathsf{view}, \beta) = 1$ iff for every round $r \geq \ell$, and each honest party i with blockchain $\mathbf{B}$ at round r, it holds that $\mathsf{u\text{-}stakefrac}(\mathbf{B}, \ell) \geq \beta$.
- $\mathsf{bd\text{-}stake\text{-}fork}^{(\ell_1, \ell_2)}(\mathsf{view}, \alpha) = 1$ iff for all rounds $r \geq \tilde{r}$, each honest party i with blockchain $\mathbf{B}$ at round r, each corrupt party j with blockchain $\tilde{\mathbf{B}}$ at round $\tilde{r}$, if there exists $\ell' \geq \ell_1 + \ell_2$ such that $\tilde{\mathbf{B}}^{\lceil \ell'} \preceq \mathbf{B}$ and for all $\tilde{\ell} < \ell'$, $\tilde{\mathbf{B}}^{\lceil \tilde{\ell}} \not\preceq \mathbf{B}$, then $\mathsf{u\text{-}stakefrac}(\tilde{\mathbf{B}}, \ell' - \ell_1) \leq \alpha$.

Based on the consistency and sufficient stake contribution predicates, we define the corresponding blockchain properties.

Definition 4 (Chain Consistency). *A blockchain protocol BLC_V satisfies ℓ_0-consistency for adversary $\mathcal{A}$ in environment $\mathcal{Z}$, if for every $\ell > \ell_0$*

$$\Pr\left[\mathsf{consistent}^\ell(\mathsf{view}) \;\middle|\; \mathsf{view} \leftarrow \mathsf{EXEC}^{\mathsf{BLC}_V}[\mathcal{A}, \mathcal{Z}, 1^\lambda]\right] \geq 1 - neg(\lambda)$$

Definition 5 (Sufficient Stake Contribution). *A blockchain protocol BLC_V satisfies (ℓ_0, β)-sufficient stake contribution for adversary $\mathcal{A}$ in environment $\mathcal{Z}$, if for every $\ell > \ell_0$*

$$\Pr\left[\mathsf{suf\text{-}stake}^\ell(\mathsf{view}, \beta) \;\middle|\; \mathsf{view} \leftarrow \mathsf{EXEC}^{\mathsf{BLC}_V}[\mathcal{A}, \mathcal{Z}, 1^\lambda]\right] \geq 1 - neg(\lambda)$$

Lastly, we consider a property called distinguishable forking which requires that sufficient stake contribution and bounded stake forking properties hold simultaneously. Note that when this is the case (and $\alpha < \beta$), it is possible to distinguish an honestly created extension of the blockchain from an adversarially created fork by examining the unique stake fraction shown in the extension or fork.

Definition 6 (Distinguishable Forking). *A blockchain protocol* BLC_V *satisfies* $(\alpha, \beta, \ell_1, \ell_2)$*-distinguishable forking for adversary* $\mathcal{A}$ *in environment* $\mathcal{Z}$*, if for every* $\ell > \ell_1$ *and* $\tilde{\ell} \geq \ell_2$

$$\Pr\left[\begin{array}{c|c} \alpha + neg_1(\lambda) < \beta \wedge & \\ \mathsf{suf\text{-}stake}^{\tilde{\ell}}(\mathsf{view}, \beta) = 1 \wedge & \mathsf{view} \leftarrow \mathsf{EXEC}^{\mathsf{BLC}_V}[\mathcal{A}, \mathcal{Z}, 1^\lambda] \\ \mathsf{bd\text{-}stake\text{-}fork}^{(\ell,\tilde{\ell})}(\mathsf{view}, \alpha + neg_1(\lambda)) = 1 & \end{array}\right] \geq 1 - neg_2(\lambda)$$

Goyal *et al.* showed in [GG17] that *Snowwhite*, a Proof-of-Stake based blockchain protocol proposed by Daian *et al.* [DPS19], satisfies all of the above properties.

Proof-of-Work Blockchain. The above properties, which will be used as a basis for the security of our construction, are all stated with respect to a blockchain based on Proof-of-Stake. It might be considered whether it would be possible to instead rely on a blockchain based on *Proof-of-Work* in which the blockchain is extended by miners solving computational puzzles (i.e. relying on the computational power of the miners). This, however, seems difficult. More specifically, in the typical Proof-of-Work setting, an adversary can locally compute a valid fork in realistic time by solving the required puzzles and ignoring input from honest miners. This would break the distinguishable forking property which our construction crucially depends on. In contrast, this property can be achieved in a Proof-of-Stake blockchain because we assume, as in [GG17], that the adversary controls only a minority stake and cannot forge digital signatures of other miners controlling a majority stake.

Additional Blockchain Notation. Each block of a blockchain $\mathbf{B}$ contains a list of records. A record is a set of fields and a field is an arbitrary string. We denote the i-th block of $\mathbf{B}$ as $\mathbf{B}_{[i]}$, the number of records in the i-th block as $|\mathbf{B}_{[i]}|$, the j-th record in the i-th block as $\mathbf{B}_{[i][j]}$, and each field in a record r as $r.\mathsf{name}$. We use the notation $r \in \mathbf{B}$ if there exists i, j such that $\mathbf{B}_{[i][j]} = r$, and $r \notin \mathbf{B}$ when this is not the case.

Also, we overload the consistency predicate, and define $\mathsf{consistent}^{\ell}(\mathbf{B}_{\mathsf{prefix}}, \mathbf{B})$ to hold for two sequences of blocks, $\mathbf{B}_{\mathsf{prefix}}$ and $\mathbf{B}$, if and only if $\mathbf{B}_{\mathsf{prefix}}^{\lceil \ell} \preceq \mathbf{B}$ i.e. $\mathbf{B}_{\mathsf{prefix}}$ with the last ℓ blocks truncated is a prefix of $\mathbf{B}$. Finally, for a blockchain satisfying $(\alpha, \beta, \ell_1, \ell_2)$-distinguishable forking, we introduce a predicate $\mathsf{ext\text{-}suf\text{-}stk}^{(\beta,\ell_1,\ell_2)}(\mathbf{B}, i)$ (short for "extended with sufficient stake"), which

takes a sequence of blocks $\mathbf{B}$ and index i where $i \geq 0$, and holds if and only if the number of blocks after the i-th block is larger than $\ell_1 + \ell_2$ and at least β fraction of stake is proved in the last ℓ_2 blocks. Intuitively, ext-suf-stk determines whether the i-th block looks honestly created, assuming stakes of adversaries are bounded by α (where $\alpha < \beta$).

3 Forward-Secure PKE without Key Update

In this section, we give definitions and the construction of our forward-secure PKE scheme without key update. In contrast to existing forward-secure PKE schemes [CHK03, GM15], both pk and sk are immutable, and since we don't employ key update to achieve forward security, the syntax looks much closer to the traditional non-forward-secure PKE schemes except we allow the encryption and decryption algorithm to make use of a blockchain protocol.

Specifically, we assume that both encryptor and decryptor are participants in a blockchain protocol, and will allow the encryption and decryption algorithms direct access to the state of the encryptor and decryptor, respectively. Note that this does not necessarily require that the encryptor or decryptor have any stake in the blockchain, but that they have the ability to broadcast messages across the blockchain network, and can extract, from their local state, their current view of the blockchain. It is assumed that both encryptor and decryptor will maintain their state by running UpdateState of the blockchain protocol, and that the encryption and decryption algorithms will have access to the most recent state when extracting the current view of the blockchain via GetRecords. In other words, we treat the input state st to the encryption and decryption algorithms as a reference to the most current state (as opposed to the value of the state at the time the algorithms are called), which will allow, for example, the algorithms to broadcast a message, and wait for this message to be included in the blockchain, before continuing execution.

The syntax of our forward-secure PKE scheme is as follows:

Definition 7 (FSPKE). *A forward-secure public-key encryption scheme without key update under the existence of a blockchain protocol* $\mathsf{BLC}_{\mathsf{V}}$ *is a tuple of algorithms* (FSPKE.KGen, FSPKE.Enc, FSPKE.Dec).

- $(pk, sk) \leftarrow \mathsf{FSPKE.KGen}(1^\lambda)$*: The key generation algorithm* FSPKE.KGen *takes as input the security parameter* λ*. It outputs a key pair* (pk, sk)*.*
- $c \leftarrow \mathsf{FSPKE.Enc}(\mathsf{st}, pk, m)$*: The encryption algorithm* FSPKE.Enc *takes as input a reference to a blockchain state* st*, a public key* pk *and a message* m*. It outputs a ciphertext* c*.*
- $m/\perp \leftarrow \mathsf{FSPKE.Dec}(\mathsf{st}, sk, c)$*: The decryption algorithm* FSPKE.Dec *takes as input a reference to a blockchain state* st*, a secret key* sk *and a ciphertext* c*. It outputs a message* m *or the symbol* $\perp$*.*

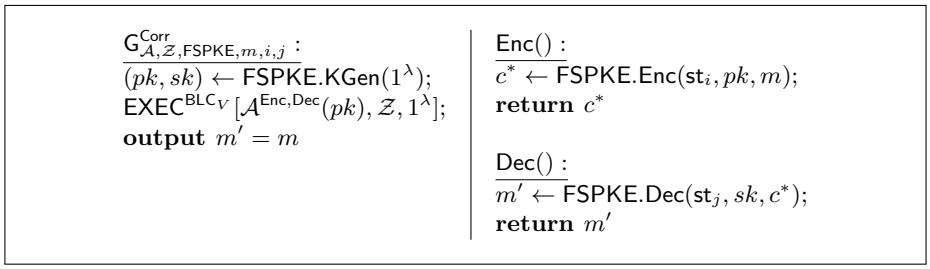

Fig. 1. Game defining correctness.

3.1 Correctness

Unlike ordinary (forward-secure) PKE, the correctness of a PKE scheme dependent on a blockchain is non-trivial. Specifically, when decryption is dependent on information obtained from or posted to the blockchain, we need to consider potential adversarial interference from other entities with access to the blockchain. Firstly, malicious miners can potentially prevent correct decryption by simply not including any information required for decryption in the blockchain. Secondly, since the basic premise of the use of the blockchain is that anyone can post a block, and by doing so, any malicious user might be able to interfere with the decryption by honest users. We capture this aspect of the use of a blockchain, by considering a correctness definition similar to a security game, in which the adversary attempts to prevent decryption of an honestly constructed ciphertext. Note that besides controlling corrupt parties, the adversary in our definition can make honestly mined blocks contain maliciously generated messages by simply broadcasting these, since we assume that all honest miners will include messages received via the broadcast functionality of the blockchain.

We define correctness via the security game shown in Figure 1 in which the adversary can instruct two honest users to encrypt and decrypt any time during the execution of the blockchain protocol via the Enc and Dec oracles. Note that for the correctness definition to be meaningful, we will only consider adversaries that query these oracles once in that order. We refer to such adversaries as *correctness-admissible*. Furthermore, note that additional restrictions on the adversary and the execution of the blockchain are likely to be required for correctness to hold for any scheme that makes meaningful use of the blockchain. In particular, the delay an adversary might introduce for messages sent to honest parties might have to be limited, and the execution of the blockchain protocol might be required to extend the blockchain. However, we will not include such restrictions or guarantees in the definition below, but introduce appropriate assumptions when showing correctness of our concrete scheme.

Definition 8 (Correctness). *We say that* FSPKE *with access to blockchain protocol* BLC_V *satisfies correctness for adversary* $\mathcal{A}$ *in environment* $\mathcal{Z}$ *if for every plaintext* m*, every pair of honest users* i *and* j *in* $\mathcal{Z}$*, there exists a negligible*

function $\varepsilon(\cdot)$ such that the following holds:

$$\Pr\left[\ \mathsf{G}^{\mathsf{Corr}}_{\mathcal{A},\mathcal{Z},\mathsf{FSPKE},m,i,j} = 1\ \right] \geq 1 - \varepsilon(\lambda)$$

3.2 Security

We will now define a security notion capturing forward security for a PKE scheme FSPKE based on a blockchain. Like in the case of correctness, the definition is non-standard due to the ability of an adversary to observe and manipulate the blockchain. Our security notion, which we denote fs-IND-CPA security, is based on the standard IND-CPA security notion for ordinary PKE, in which the adversary is challenged to distinguish between the encryption c^* of two adversarially chosen messages, m_0 and m_1. However, we allow the adversary to access two new oracles: Leak and HonestDec. The first oracle, Leak, captures the notion of a key compromise. When it's invoked, it will return the secret key to the adversary, but before doing so, it ensures that the challenge ciphertext has been decrypted by running $m^* \leftarrow \mathsf{FSPKE.Dec}(\mathsf{st}, sk, c^*)$. In previous forward security notions, this oracle would correspond to an oracle that updates the secret key and returns the new (updated) key to the adversary.

The second oracle, $\mathsf{HonestDec}(c)$, captures potential information leakage from records posted on the blockchain by honest users in the decryption process[4]. Specifically, in the blockchain setting, an honest user might be required to post information related to a ciphertext c or their secret key sk, in order to be able to decrypt c. Since the blockchain is public, an adversary will be able to obtain this information just by monitoring the blockchain. To capture this, the oracle HonestDec allows the adversary to submit any ciphertext c, which the oracle will decrypt as $m \leftarrow \mathsf{FSPKE.Dec}(\mathsf{st}, sk, c)$. However, as we consider a CPA security notion, the decryption result m will not be returned to the adversary (he will only be able to observe any information posted to the blockchain in the decryption process). Our definition can be extended to a CCA notion, simply by returning m and restricting the adversary from submitting c^*. Note that in our definition below, no restrictions are placed on c submitted to HonestDec.

Finally, note that the fs-IND-CPA definition *itself* is generic: it does not place any assumptions on the adversary in terms of adversarial control of the blockchain (e.g. the amount of stake held by the adversary). For our concrete scheme, which will be presented in Section 3.3, we will show that fs-IND-CPA security holds, assuming the stake controlled by the adversary is sufficiently small as in [GG17].

Security is defined via the game shown in Figure 2. We say that an adversary fs-IND-CPA $\mathcal{A}$ is *admissible* if $\mathcal{A}$ queries the challenge oracle Chal once with messages m_0 and m_1 of equal length, and only queries the Leak oracle after Chal has been queried (without loss of generality, we can assume any $\mathcal{A}$ always queries both oracles).

[4] Note that encryption might likewise require information being posted to the blockchain, but this is already captured by running the encryption algorithm when constructing the challenge ciphertext c^*.

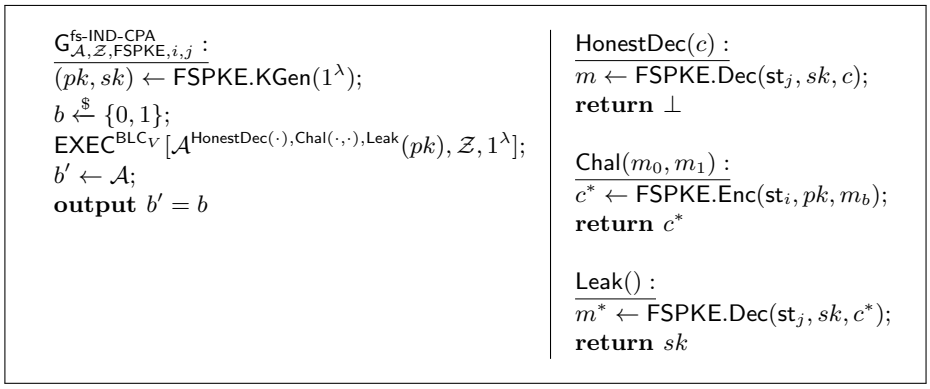

Fig. 2. Security game defining fs-IND-CPA security.

Definition 9 (fs-IND-CPA). *Let* BLC_V *be a blockchain protocol with the validity predicate* V*, and let* $\mathsf{FSPKE} = (\mathsf{FSPKE.KGen}, \mathsf{FSPKE.Enc}, \mathsf{FSPKE.Dec})$ *be a public-key encryption scheme with access to* BLC_V*. We define the advantage* $\mathsf{Adv}^{\mathsf{fs\text{-}IND\text{-}CPA}}_{\mathcal{A},\mathsf{FSPKE}}(\lambda)$ *of an adversary* $\mathcal{A}$ *against the fs-IND-CPA security of* FSPKE *as*

$$\mathsf{Adv}^{\mathsf{fs\text{-}IND\text{-}CPA}}_{\mathcal{A},\mathsf{FSPKE}}(\lambda) \coloneqq \left| \Pr\left[\mathsf{G}^{\mathsf{fs\text{-}IND\text{-}CPA}}_{\mathcal{A},\mathcal{Z},\mathsf{FSPKE},i,j} = 1\right] - \frac{1}{2} \right|$$

where the security game $\mathsf{G}^{\mathsf{fs\text{-}IND\text{-}CPA}}_{\mathcal{A},\mathcal{Z},\mathsf{FSPKE},i,j}$ *is defined in Figure 2. We say that* FSPKE *is fs-IND-CPA secure against an admissible adversary* $\mathcal{A}$ *in environment* $\mathcal{Z}$ *if for all honest users* i *and* j *in* $\mathcal{Z}$*,* $\mathsf{Adv}^{\mathsf{fs\text{-}IND\text{-}CPA}}_{\mathcal{A},\mathsf{FSPKE}}(\lambda)$ *is negligible in* λ*.*

Note that similar to puncturable encryption [GM15], the above security notion guarantees fine-grained forward security i.e. the scheme must support removing the ability to decrypt just a single ciphertext. This improves upon the notion for standard forward-secure schemes based on key update, in which the ability to decrypt all ciphertexts constructed between two key updates is lost in the second key update. Note that adjusting the time period between key updates in this type of scheme is a challenging task; frequent updates implies that the ability to decrypt any ciphertext the decryptor cannot immediately access and decrypt will be lost, whereas infrequent updates implies that any adversary gaining access to the decryption key will have the ability to decrypt a potentially large number of previous ciphertexts i.e. any ciphertext constructed within the current time period (as well as future ciphertexts). In contrast, fine-grained forward security does not require a notion of time, and any ciphertext not yet decrypted by the decryptor will remain decryptable. In this sense, a fine-grained forward-secure scheme provides a functionality closer to ordinary non-forward-secure encryption, while still providing strong security guarantees in the case of key compromise. It should be noted, however, that standard fine-grained forward-secure schemes inherently do not protect against a particular

type of message suppression attack [BG20]. In Section 5, we discuss the details of this as well as how our particular construction allows this type of attack to be mitigated.

3.3 Construction

Our construction is inspired by the idea behind the construction of one-time programs using a proof-of-stake blockchain presented by Goyal *et al.* [GG17], in particular, the use of a proof-of-stake blockchain in combination with a witness encryption scheme[5]. In our construction, a message is encrypted under an NP statement requiring that a certain type of record associated to an ephemeral secret to be signed by the receiver and posted to the blockchain. Here, the signing key is the receiver's long-term secret. The decryption algorithm, which has access to the signing key, constructs and signs such a record, posts this to the blockchain, and waits until the blockchain has been sufficiently extended. Then, using the ephemeral secret and the blockchain containing the corresponding record as a witness, the decryption algorithm is able to decrypt the message.

Note that the ephemeral secret is the only secret required to construct a valid witness required for decryption as the blockchain is assumed to be public. Hence, neither the record posted to the blockchain nor a key compromise must leak this. The former is ensured by using a one-way hash function (and high-entropy ephemeral secrets), and the latter is ensured by deleting the ephemeral secret once decryption has been completed. Note also, that an attacker without access to the long-term signing key will be unable to construct an appropriate record that can be used for decryption, assuming the signature scheme is secure.

The key to making this construction forward secure is to require the NP relation to check that the record used in the witness is the *first* record in the blockchain that allows decryption. This will prevent an attacker from creating a valid witness for a given ciphertext once this has been decrypted by the receiver, even if the attacker gains access to the long-term signing key.

The above assumes that the attacker cannot manipulate the blockchain itself. To ensure the security extends to attackers with a minority stake in the blockchain, we rely on the distinguishable forking property (Definition 6). More specifically, the distinguishable forking property guarantees that honestly created blockchain extensions can be distinguished from adversarially constructed forks by examining the unique stake in blockchain. Hence, by letting the NP relation additionally check that blockchain used in the witness is of sufficient

[5] As a one-time program is a powerful primitive, it might be considered to base the construction of a forward-secure encryption scheme directly on this (besides additional appropriate primitives). However, we note that it is not clear whether such a construction will be able to meet our security notions (e.g. [GG17] does not consider correctness against malicious adversaries whereas we do), and any potential construction will be much more complicated due to the generality of one-time programs based on garbled circuits. Hence, we focus on a direct construction based on a proof-of-stake blockchain.

A pair of NP instance $x = (\mathbf{B} \,\|\, \mathsf{id} \,\|\, pk)$ and a witness $w = (\mathbf{B}' \,\|\, esk)$ satisfies R_{FSPKE} if and only if the following properties are satisfied:

$$\begin{aligned} R_{\mathsf{FSPKE}}(\mathbf{B} \,\|\, \mathsf{id} \,\|\, pk,\ \mathbf{B}' \,\|\, esk) \coloneqq\ & R_{\mathsf{ValidBlocks}}(\mathbf{B}, \mathbf{B}') \wedge R_{\mathsf{ValidEsk}}(pk, \mathsf{id}, \mathbf{B}', esk) \\ & \wedge \mathsf{consistent}^{\ell_c}(\mathbf{B}, \mathbf{B}') \end{aligned}$$

where

$$\begin{aligned} R_{\mathsf{ValidBlocks}}(\mathbf{B}, \mathbf{B}') &\coloneqq V(\mathbf{B}) = 1 \wedge V(\mathbf{B}') = 1 \\ R_{\mathsf{ValidEsk}}(pk, \mathsf{id}, \mathbf{B}', esk) &\coloneqq \exists i^*, \exists j^*, \\ &\quad \mathsf{ext\text{-}suf\text{-}stk}^{(\beta,\ell_1,\ell_2)}(\mathbf{B}', i^*) \\ &\quad \wedge R_{\mathsf{DecAttempt}}(\mathsf{id}, pk, \mathbf{B}'_{[i^*][j^*]}) \\ &\quad \wedge R_{\mathsf{NotYetDecrypted}}(\mathbf{B}', i^*, j^*, \mathsf{id}, pk) \\ &\quad \wedge R_{\mathsf{KnowsEsk}}(\mathbf{B}', i^*, j^*, esk) \\ R_{\mathsf{DecAttempt}}(\mathsf{id}, pk, r) &\coloneqq (r.\mathsf{id} = \mathsf{id}) \wedge (\mathsf{Sig.Ver}(pk, r.\mathsf{id} \,\|\, r.\sigma, r.\mathsf{cert}) = 1) \\ R_{\mathsf{NotYetDecrypted}}(\mathbf{B}', i^*, j^*, \mathsf{id}, pk) &\coloneqq (\forall 0 \le j < j^*, \neg R_{\mathsf{DecAttempt}}(\mathsf{id}, pk, \mathbf{B}'_{[i^*][j]})) \\ &\quad \wedge (\forall 0 \le i < i^*, \forall 0 \le j < |\mathbf{B}'_{[i]}|, \\ &\quad\quad \neg R_{\mathsf{DecAttempt}}(\mathsf{id}, pk, \mathbf{B}'_{[i][j]})) \\ R_{\mathsf{KnowsEsk}}(\mathbf{B}', i^*, j^*, esk) &\coloneqq H(esk) = \mathbf{B}'_{[i^*][j^*]}.\sigma \end{aligned}$$

Fig. 3. An NP relation R_{FSPKE} based on the blockchain protocol BLC_V with validity predicate V and parameters $\mathsf{par} = (\beta, \ell_c, \ell_1, \ell_2)$, Sig is a public key signature scheme, and H is a one-way hash function.

length and has sufficient stake, we can ensure that the attacker cannot decrypt by constructing a fork of the blockchain.

Let WE be a witness encryption scheme for the NP relation R_{FSPKE} (defined in Figure 3), BLC_V a blockchain protocol with the validity predicate V, Sig a public key signature scheme, and H a one-way hash function. We present our construction, FSPKE, of a forward-secure public-key encryption scheme without key update in Figure 4. Note that the scheme depends on a set of parameters $\mathsf{par} = (\beta, \ell_c, \ell_1, \ell_2)$ which should be set according to the properties of the underlying blockchain protocol.

On the Relation R_{FSPKE}. The relation R_{FSPKE} used in WE and defined in Figure 3 makes use of several sub-relations. We discuss the intuition of these in the following. $R_{\mathsf{ValidBlocks}}$ ensures that both sequences of blocks satisfy blockchain-protocol-specific requirements i.e. it denies malformed inputs. R_{ValidEsk} ensures that the ciphertext has not yet been decrypted. It requires that the given esk is valid for the *first* record on the blockchain which satisfies $R_{\mathsf{DecAttempt}}$. $\mathsf{ext\text{-}suf\text{-}stk}$ used in R_{ValidEsk} ensures the i^*-th block is honestly created with all but negligible

$\underline{(pk, sk) \leftarrow \mathsf{FSPKE.KGen}(1^\lambda)}$

1. $(pk, sk) \leftarrow \mathsf{Sig.KGen}(1^\lambda)$.
2. Output (pk, sk).

$\underline{c \leftarrow \mathsf{FSPKE.Enc}(\mathsf{st}, pk, m)}$

1. $\mathbf{B} \leftarrow \mathsf{BLC}_V.\mathsf{GetRecords}(1^\lambda, \mathsf{st})$.
2. $\mathsf{id} \xleftarrow{\$} \{0,1\}^\lambda$.
3. $x \coloneqq (\mathbf{B} \,\|\, \mathsf{id} \,\|\, pk)$.
4. $CT \leftarrow \mathsf{WE.Enc}(1^\lambda, x, m)$.
5. Output $c \coloneqq (\mathsf{id}, CT)$.

$\underline{m \leftarrow \mathsf{FSPKE.Dec}(\mathsf{st}, sk, c)}$

1. Parse c as (id, CT).
2. $esk \xleftarrow{\$} \{0,1\}^\lambda$.
3. $\sigma \leftarrow H(esk)$.
4. $\mathsf{cert} \leftarrow \mathsf{Sig.Sign}(sk, \mathsf{id} \,\|\, \sigma)$.
5. $r \coloneqq (\mathsf{id} \,\|\, \sigma \,\|\, \mathsf{cert})$.
6. $\mathsf{BLC}_V.\mathsf{Broadcast}(1^\lambda, r)$.
7. $\mathbf{B} \leftarrow \mathsf{BLC}_V.\mathsf{GetRecords}(1^\lambda, \mathsf{st})$.
8. While $r \notin \mathbf{B}^{\lceil(\ell_1+\ell_2)}$:
9. $\quad \mathbf{B} \leftarrow \mathsf{BLC}_V.\mathsf{GetRecords}(1^\lambda, \mathsf{st})$.
10. $w \coloneqq (\mathbf{B} \,\|\, esk)$.
11. $m \leftarrow \mathsf{WE.Dec}(CT, w)$.
12. Erase esk and then output m.

Fig. 4. A construction of FSPKE where WE is a witness encryption scheme for the NP relation R_{FSPKE} (defined in Figure 3), BLC_V is a blockchain protocol with the validity predicate V and parameters $\mathsf{par} = (\beta, \ell_c, \ell_1, \ell_2)$, Sig is a public key signature scheme, and H is a one-way hash function.

probability. $R_{\mathsf{DecAttempt}}$ is true if the given record r contains a decryption attempt for the ciphertext associated with id. $R_{\mathsf{NotYetDecrypted}}$ ensures that before the j^*-th record in the i^*-th block in $\mathbf{B}'$, there're no valid decryption attempts for the ciphertext associated with id. This relation guarantees that the ciphertext can be decrypted only once. R_{KnowsEsk} ensures that the party who is trying to decrypt knows the ephemeral secret key esk for the first decryption attempt.

3.4 Proof of Correctness

Before showing correctness of our scheme, we will introduce mild assumptions regarding the execution of the blockchain. Firstly, we will restrict our attention to blockchain executions that lead to a sufficient growth of the blockchain. More specifically, we will refer to a blockchain execution as *ℓ-growth respecting* if the blockchain of all honest parties is extended with at least ℓ blocks following a broadcast by an honest party. Finally, we restrict the delay in terms of growth of the blockchain, an adversary might introduce for messages broadcast by honest parties. Specifically, we refer to a blockchain execution as *$\tilde{\ell}$-delay respecting*, if the blockchain of any honest users is extended with at most $\tilde{\ell}$ blocks between an honest user broadcasting a message and this is delivered to all other honest users.

Theorem 1. *Assume the signature scheme* Sig *is EUF-CMA secure and that the blockchain protocol* BLC_V *provides* ℓ_c*-consistency, and* (ℓ_2, β)*-sufficient stake for*

all PPT adversaries with stake at most α' in environment $\mathcal{Z}$. Then the construction described in Figure 4 with parameters $\mathsf{par} = (\beta, \ell_c, \ell_1, \ell_2)$ *satisfies correctness for any PPT correctness-admissible adversary $\mathcal{A}$ in $\mathcal{Z}$ with stake at most $\alpha < \mathsf{min}(\alpha', \beta)$ in blockchain executions that are $\tilde{\ell}$-delay and $(\tilde{\ell}+\ell_1+2\ell_2)$-growth respecting.*

Proof (Theorem 1). Firstly note that the definition of ℓ_c-consistency directly implies that for blockchain $\mathbf{B}$ used in the encryption performed in the Enc oracle and the blockchain $\mathbf{B}'$ used in the decryption in the Dec oracle, $\mathsf{consistent}^{\ell_c}(\mathbf{B}, \mathbf{B}')$ holds with overwhelming probability.

Secondly, since the execution is $(\tilde{\ell}+\ell_1+2\ell_2)$-growth respecting, the blockchain $\mathbf{B}'$ contained in st_j used in the decryption must be extended with $\tilde{\ell} + \ell_1 + 2\ell_2$ blocks after the broadcast of r in line 6 of the decryption algorithm. Since the execution is also $\tilde{\ell}$-delay respecting, r must have been delivered to all honest miners before $\mathbf{B}'$ has been extended with $\tilde{\ell}$ blocks, and due to the (ℓ_2, β)-sufficient stake property and $\alpha < \beta$, the next ℓ_2 blocks must contain an honestly mined block (which must include r unless r has already been posted) with overwhelming probability. Hence, there must be at least $\ell_1+\ell_2$ blocks after the block containing r, and again due to the (ℓ_2, β)-sufficient stake property, the ℓ_2 last blocks of these will have stake at least β. This implies that $\mathsf{ext\text{-}suf\text{-}stk}^{(\beta,\ell_1,\ell_2)}(\mathbf{B}', i^*)$ is satisfied, where i^* is the index of the block containing r.

Combined with the observation that r is honestly constructed, the above implies that the witness $\mathbf{B}' \,\|\, esk$ constructed in the decryption is a valid witness *unless* $R_{\mathsf{NotYetDecrypted}}$ does not hold. This happens only if $\mathbf{B}'$ contains a block with index less than i^* with a record r' for which $r'.\sigma \neq r.\sigma$ but which satisfies $R_{\mathsf{DecAttempt}}(\mathsf{id}, pk, r')$ for the id used in the encryption. This in turn implies that $r'.\mathsf{cert}$ is a valid signature on $r'.\mathsf{id} \,\|\, r'.\sigma$. However, if $\mathcal{A}$ can cause such a record to be added to $\mathbf{B}'$, we can construct a PPT algorithm $\mathcal{B}$ which breaks the EUF-CMA of the digital signature scheme. $\mathcal{B}$ simply plays the correctness game with $\mathcal{A}$ simulating all honest parties, and using his signing oracle to obtain $r.\mathsf{cert}$ corresponding to a signature on $r.\mathsf{id} \,\|\, r.\sigma$. After the game finishes, it searches $\mathbf{B}' \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st}_j)$ for a valid record r' (posted by $\mathcal{A}$) such that $\mathsf{Sig.Ver}(pk, r'.\mathsf{id} \,\|\, r'.\sigma, r'.\mathsf{cert}) = 1$ holds. Lastly, it outputs the pair $(r'.\mathsf{id} \,\|\, r'.\sigma, r'.\mathsf{cert})$ in the EUF-CMA security game.

Since the signature scheme is assumed to be secure, we conclude that $\mathcal{B}$ will only succeed with negligible probability, and hence, that $R_{\mathsf{NotYetDecrypted}}$ will hold with overwhelming probability. Thus the theorem holds. (Theorem 1) □

Note that in the above, we assume that WE does not impose a length bound on the used witness. If the maximum witness length of the witness encryption is bounded, we additionally need to assume that the number of records posted to the blockchain by $\mathcal{A}$ for a certain period is bounded for correctness to hold. In other words, we would require the honest user is able to decrypt before $\mathcal{A}$ posts so many blocks to the blockchain such that it cannot be used as a witness due to the length bound being exceeded. A similar assumption is necessary in the framework of [GG17].

Lastly, we note that correctness would still hold even if the encryptor bases his encryption on a previously obtained version of the blockchain as opposed to the most recent up-to-date version. This is because our construction (Figure 3) only requires the blockchain $\mathbf{B}$ used in encryption to be a prefix of and be consistent (w.r.t. $\mathsf{consistent}^{\ell}$ as defined in Section 2.2) with the decryptor's blockchain $\mathbf{B}'$. However, note again that if the witness encryption only supports witnesses of bounded size, the difference in terms of blocks between the versions of the blockchain used by encryptor and decryptor cannot exceed this bound, as decryption would otherwise fail[6].

3.5 Efficiency

The efficiency of our construction essentially follows from the efficiency of the underlying signature scheme, witness encryption scheme, and blockchain. We emphasize that neither encryptor nor decryptor are required to participate in the blockchain protocol itself, but are only required to be able to access an up-to-date version of the blockchain, and in case of the decryptor, be able to post a message to the blockchain e.g. by requesting a miner to do so. Depending on the premise of the blockchain protocol execution, the latter might involve an additional cost to the decryptor (e.g. paying a fee to the miner).

In more detail, key generation and public/private key size correspond to that of the signature scheme, and the computational encryption cost and the ciphertext size correspond to that of the witness encryption scheme, assuming accessing the blockchain does not involve any computational requirements. Decryption firstly requires the decryptor to post a signed message to the blockchain. Note that he will not be able to immediately decrypt once this has been posted, but must wait for the blockchain to grow sufficiently to satisfy distinguishable forking (Definition 6). Once this happens, he will invoke the decryption of the witness encryption scheme, which will most likely dominate the computational decryption cost (compared to signing). We refer the reader to Section 2.1 for a discussion of potential witness encryption instantiations.

Finally, we note that the relation in Figure 3, which is required to be implemented by the witness encryption, is relatively complex, which could be an efficiency concern as ciphertext size and encryption/decryption cost typically scale with the size and complexity of the encryption statement and witness[7][8]. However, as noted in [LJKW18], this can be addressed by the use of succinct

[6] For ease of notation, as in [GG17], we use the entire blockchain $\mathbf{B}'$ as part of the witness w. However, we note that essentially only the blocks appended to the blockchain after encryption suffice as part of the witness w.

[7] Note that in our construction, the lower bound of the witness size would be $\ell_1+\ell_2+\ell_d$ blocks where ℓ_1 and ℓ_2 are blockchain-specific parameters from the distinguishable forking property (see Section 2.2), and ℓ_d is the difference in the number of blocks between the blockchain obtained by encryptor and decryptor.

[8] We note the approach by Goyal *et al.* [GKM+20] allows efficient encryption and only requires the decryptor to perform a potentially heavy computation related to the relevant statement and witness.

non-interactive arguments of knowledge (SNARKs) (e.g. see [Gro16]). In our construction, the decryptor could include a SNARK common reference string in his public key[9], allowing the relation in Figure 3 to be proved using the SNARK and the witness encryption to only rely on the verification and succinct witness from the SNARK. This would alleviate concerns regarding encryption cost and ciphertext size.

4 Security Analysis

The following theorem establishes the security of our construction.

Theorem 2. *Assume* WE *is an extractable witness encryption scheme for the NP relation* R_{FSPKE}, Sig *is an EUF-CMA-secure signature scheme,* H *is a one-way hash function, and* BLC_V *is a blockchain protocol satisfying* $(\alpha, \beta, \ell_1, \ell_2)$*-distinguishable forking property for any PPT adversary with stake fraction at most* α *in environment* $\mathcal{Z}$*. Then the construction described in Figure 4 is fs-IND-CPA secure for any admissible PPT adversary* $\mathcal{A}$ *in* $\mathcal{Z}$ *with at most* α *stake fraction.*

4.1 Proof of Theorem 2

Simulation of the blockchain. Theorem 2 is with respect to an adversary $\mathcal{A}$ who controls at most an α stake fraction of the blockchain. With the exception of Claim 3, our security reduction will simulate the parties holding the remaining stake fraction for $\mathcal{A}$, by honestly executing the blockchain protocol BLC_V. We do not include an explicit simulation of this in the following proof.

Proof (Theorem 2). Let FORGE be the event that $\mathcal{A}$ causes honest user j to add a maliciously constructed record r^* to the blockchain contained in st_j that can be used to decrypt the challenge ciphertext c^*. More precisely, FORGE denotes that r^* is the first record in sequence of blocks $\mathbf{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st}_j)$ satisfying $\mathsf{Sig.Ver}(pk, r^*.\mathsf{id} \,\|\, r^*.\sigma, r^*.\mathsf{cert}) = 1$. Note that since the Leak oracle will add a valid record to the blockchain contained in st_j for decryption of c^*, $\mathcal{A}$ needs to compute $r^*.\mathsf{cert}$ and post r^* *before* Leak does so (i.e. without sk) for FORGE to occur. That is, intuitively speaking, posting r^* means that $\mathcal{A}$ can forge a valid signature $r^*.\mathsf{cert}$. In the following lemma, we formalize this intuition.

Lemma 1. *Assume that* Sig *is an EUF-CMA secure signature scheme. Then* $\Pr[\mathsf{FORGE}] < neg(\lambda)$.

Proof (Lemma 1). If FORGE occurs, we can construct an adversary $\mathcal{B}_{\mathsf{Sig}}$ which breaks EUF-CMA security of Sig. $\mathcal{B}_{\mathsf{Sig}}$ simulates the role of a challenger in the fs-IND-CPA game for $\mathcal{A}$, and is defined as follows:

[9] To maintain forward security, the randomness and trapdoor for this common reference string must be securely erased by the key pair holder after key generation.

1. Upon receiving pk in the EUF-CMA game, $\mathcal{B}_{\mathsf{Sig}}$ forwards pk to $\mathcal{A}$. When running, $\mathcal{B}_{\mathsf{Sig}}$ simulates all honest parties in the blockchain and executes the blockchain protocols honestly. If $\mathsf{HonestDec}(c)$ is called where $c = (\mathsf{id}, CT)$, $\mathcal{B}_{\mathsf{Sig}}$ performs the decryption operations as described in Figure 4 except it computes cert using the signing oracle of the EUF-CMA game. Lastly, $\mathcal{B}_{\mathsf{Sig}}$ adds $(\mathsf{id} \,\|\, \sigma)$ to a set Σ. When $\mathcal{A}$ calls Leak, $\mathcal{B}_{\mathsf{Sig}}$ aborts $\mathcal{A}$ after decryption of c^*, before the secret key sk is returned.
2. When $\mathcal{A}$ finishes its execution (or is terminated by $\mathcal{B}_{\mathsf{Sig}}$ due to a call to Leak), $\mathcal{B}_{\mathsf{Sig}}$ searches the blockchain $\mathbf{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st}_j)$ for a valid record r^* such that $(r^*.\mathsf{id} \,\|\, r^*.\sigma) \notin \Sigma$, and outputs $(r^*.\mathsf{id} \,\|\, r^*.\sigma, r^*.\mathsf{cert})$ if such a record is found.

From the above description, it should be clear that $\mathcal{B}_{\mathsf{Sig}}$ provides a perfect simulation for $\mathcal{A}$ up until abortion, and that, assuming FORGE occurs, $\mathcal{B}_{\mathsf{Sig}}$ returns a valid forgery. (Lemma 1) □

Let S be the event that the adversary $\mathcal{A}$ wins the fs-IND-CPA game. In the following lemma, we consider the case $\mathcal{A}$ wins the game without causing a valid maliciously constructed record to the blockchain of honest user j that can be used for decrypting the challenge ciphertext, i.e. without FORGE occurring.

Lemma 2. *Assume* WE *is an extractable witness encryption scheme, H is a one-way hash function, and the blockchain protocol* BLC_V *satisfies* $(\alpha, \beta, \ell_1, \ell_2)$*-distinguishable forking property in* $\mathcal{Z}$*. Then* $\left|\Pr[S|\neg\mathsf{FORGE}] - \frac{1}{2}\right| \leq neg(\lambda)$

Proof (Lemma 2). Assume there exists an fs-IND-CPA attacker $\mathcal{A}$ with non-negligible advantage $\varepsilon = \left|\Pr[S|\neg\mathsf{FORGE}] - \frac{1}{2}\right|$. From $\mathcal{A}$, we construct an attacker $\mathcal{A}^{\mathsf{WE}}$ against WE as follows. Firstly, we choose random $esk \leftarrow \{0,1\}^n$, and compute $y \leftarrow H(esk)$. The value y will be hardcoded into $\mathcal{A}^{\mathsf{WE}}$, and we use the notation $\mathcal{A}^{\mathsf{WE}}_y$ to denote this. Hardcoding y into $\mathcal{A}^{\mathsf{WE}}$ is needed, as below, we will consider a value y given by an external one-way challenger for H, and hence, $\mathcal{A}^{\mathsf{WE}}$ cannot generate y internally.

$\mathcal{A}^{\mathsf{WE}}_y$ will simulate the fs-IND-CPA game for $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as follows:

1. $\mathcal{A}^{\mathsf{WE}}_y$ generates a FSPKE key pair as $(pk, sk) \leftarrow \mathsf{FSPKE.KGen}(1^\lambda)$.
2. $\mathcal{A}^{\mathsf{WE}}_y$ sends pk to $\mathcal{A}_1$ and forwards its output m_0, m_1 and $x = (\mathbf{B} \,\|\, \mathsf{id} \,\|\, pk)$ as the challenge instance in the extractability game, where $\mathbf{B}$ and id are computed as $\mathbf{B} \leftarrow \mathsf{BLC}_V.\mathsf{GetRecords}(1^\lambda, \mathsf{st}_i)$ and $\mathsf{id} \xleftarrow{\$} \{0,1\}^\lambda$ respectively.
3. Upon receiving the challenge ciphertext c^*, $\mathcal{A}^{\mathsf{WE}}_y$ forwards c^* to $\mathcal{A}_2$.
   - If $\mathsf{HonestDec}$ oracle is called by $\mathcal{A}$, $\mathcal{A}^{\mathsf{WE}}_y$ executes step 1 to 5 of the decryption algorithm for the given ciphertext c, as defined in the construction, except it replaces σ with the hardcoded value y if c is the challenge ciphertext[10].

[10] Note that $\mathcal{A}^{\mathsf{WE}}_y$ cannot fully decrypt the challenge ciphertext c^*, as it does not know the preimage of the hardcoded value y, which is required to construct a witness for decryption.

- If the Leak oracle is called by $\mathcal{A}$, $\mathcal{A}_y^{\mathsf{WE}}$ responds in the same way as in $\mathsf{HonestDec}$, and then returns sk.

4. Lastly, $\mathcal{A}_2$ outputs b, and $\mathcal{A}_y^{\mathsf{WE}}$ forwards this as its own response in the extractability game.

From the above description, it should be clear that the view of $\mathcal{A}$ is identical to the fs-IND-CPA game, and that if $\mathcal{A}$ successfully distinguishes the encryption of m_0 and m_1, so will $\mathcal{A}_y^{\mathsf{WE}}$ in the extractable witness encryption game. Since WE is extractable, there exists a PPT extractor $\mathcal{E}$ for $\mathcal{A}_y^{\mathsf{WE}}$, and assuming we can show that $\mathcal{A}_y^{\mathsf{WE}}$ successfully distinguishes with a non-negligible advantage, $\mathcal{E}$ will likewise be able to compute a valid witness with non-negligible advantage. However, here a subtle issue arises: from the assumption that the advantage of $\mathcal{A}$ is ε, it only follows that $\mathcal{A}_y^{\mathsf{WE}}$ has advantage ε when the choice of y is considered part of the probability space defining the advantage. For a fixed value of y, even if this is correctly distributed, we can no longer draw the conclusion that $\mathcal{A}_y^{\mathsf{WE}}$ has advantage ε. Nevertheless, the following claim shows that, with probability $\varepsilon/2$ over the choice of y, $\mathcal{A}_y^{\mathsf{WE}}$ will have an advantage larger than $\varepsilon/2$.

Claim 1 *Let b' denote the bit output by $\mathcal{A}_y^{\mathsf{WE}}$, let b denote the challenge bit in the extractability game, and let Good_y denote the event that $\Pr[b = b'] \geq \varepsilon/2 + 1/2$. Then, $\Pr[\mathsf{Good}_y] \geq \varepsilon/2$, where the probability is taken over a random choice of $esk \leftarrow \{0,1\}^n$ and $y \leftarrow H(esk)$.*

Proof (Claim 1). Let Succ denote the event $b' = b$ when $esk \leftarrow \{0,1\}^n$ is picked at random and $y \leftarrow H(esk)$. From the construction of $\mathcal{A}_y^{\mathsf{WE}}$ and the assumption that the advantage of $\mathcal{A}$ is ε, we have that $\Pr[\mathsf{Succ}] = \varepsilon + 1/2$. Hence,

$$\begin{aligned}
\varepsilon + \frac{1}{2} &= \Pr[\mathsf{Succ}|\mathsf{Good}_y]\Pr[\mathsf{Good}_y] + \Pr[\mathsf{Succ}|\neg\mathsf{Good}_y]\Pr[\neg\mathsf{Good}_y] \\
&\leq \Pr[\mathsf{Good}_y] + \Pr[\mathsf{Succ}|\neg\mathsf{Good}_y] \\
&\leq \Pr[\mathsf{Good}_y] + \frac{\varepsilon}{2} + \frac{1}{2}
\end{aligned}$$

where the last inequality follows by the definition of $\neg\mathsf{Good}_y$. Rearranging the terms, we obtain $\Pr[\mathsf{Good}_y] \geq \varepsilon/2$. (Claim 1) □

The above claim allows us to conclude that we can extract a valid witness for x specified by $\mathcal{A}_y^{\mathsf{WE}}$ (including its internal fs-IND-CPA attacker $\mathcal{A}$) with non-negligible probability, despite invoking the extractor $\mathcal{E}$ with $\mathcal{A}_y^{\mathsf{WE}}$ for a fixed (but randomly chosen) y. This can be seen as follows. Let w denote the witness extracted by $\mathcal{E}$ from $\mathcal{A}_y^{\mathsf{WE}}$. Then we have that

$$\begin{aligned}
\Pr[R_{\mathsf{FSPKE}}(x,w)] &\geq \Pr[b = b' \wedge R_{\mathsf{FSPKE}}(x,w)] \\
&\geq \Pr[\mathsf{Good}_y] \cdot \Pr[b = b' \wedge R_{\mathsf{FSPKE}}(x,w)|\mathsf{Good}_y] \\
&\geq \varepsilon/2 \cdot \Pr[b = b' \wedge R_{\mathsf{FSPKE}}(x,w)|\mathsf{Good}_y]. \qquad (1)
\end{aligned}$$

By definition, Good_y ensures that the advantage of $\mathcal{A}_y^{\mathsf{WE}}$ is greater than $\varepsilon/2$, and we obtain that

$$\begin{aligned}\varepsilon/2 &\leq \Pr[b = b' | \mathsf{Good}_y] - \frac{1}{2} \\ &\leq \Pr[b = b' \wedge R_{\mathsf{FSPKE}}(x, w) | \mathsf{Good}_y] + \Pr[b = b' \wedge \neg R_{\mathsf{FSPKE}}(x, w) | \mathsf{Good}_y] - \frac{1}{2} \\ &\leq \Pr[b = b' \wedge R_{\mathsf{FSPKE}}(x, w) | \mathsf{Good}_y] + neg(\lambda)\end{aligned}$$

where the last inequality follows from the extractability of WE (note that extractability requires a successful extractor exists for all successful adversaries, including any adversary $\mathcal{A}_y^{\mathsf{WE}}$ for values of y such that Good_y is satisfied). Rearranging the terms yields that

$$\Pr\left[b = b' \wedge R_{\mathsf{FSPKE}}(x, w) | \mathsf{Good}_y\right] \geq \frac{\varepsilon}{2} - neg(\lambda)$$

and combining this with (1) we obtain that

$$\Pr\left[R_{\mathsf{FSPKE}}(x, w)\right] \geq \frac{\varepsilon}{2} \cdot (\frac{\varepsilon}{2} - neg(\lambda)).$$

Note that if ε is non-negligible, then so is $\Pr[R_{\mathsf{FSPKE}}(x, w)]$. In other words, with non-negligible probability, we obtain a valid witness w for x specified by $\mathcal{A}_y^{\mathsf{WE}}$ via the extractor $\mathcal{E}$.

In the following, we will show that if a valid witness can be extracted, we can either break the onewayness of the hash function H, or the distinguishable forking property of BLC_V.

Let HONEST be the event that $\mathcal{E}$ outputs a sequence of blocks $\mathbf{B}'$ containing the record r^* honestly constructed in the first decryption query of the challenge ciphertext c^* (either a query to Dec or Leak) as the first valid record that allows decryption of c^*. We have that

$$\begin{aligned}\Pr[R_{\mathsf{FSPKE}}(x, w)] &= \Pr[R_{\mathsf{FSPKE}}(x, w) | \mathsf{HONEST}] \cdot \Pr[\mathsf{HONEST}] \\ &\quad + \Pr[R_{\mathsf{FSPKE}}(x, w) | \neg\mathsf{HONEST}] \cdot \Pr[\neg\mathsf{HONEST}] \\ &\leq \Pr[R_{\mathsf{FSPKE}}(x, w) | \mathsf{HONEST}] \\ &\quad + \Pr[R_{\mathsf{FSPKE}}(x, w) | \neg\mathsf{HONEST}] \qquad (2)\end{aligned}$$

Claim 2 *If* $\Pr[R_{\mathsf{FSPKE}}(x, w) | \mathsf{HONEST}]$ *is non-negligible, there exists an adversary* $\mathcal{B}_{\mathsf{OW}}$ *against the onewayness of* H *with non-negligible advantage.*

Proof (Claim 2). $\mathcal{B}_{\mathsf{OW}}$ is constructed as follows. Given a challenge y^*, $\mathcal{B}_{\mathsf{OW}}$ simply constructs $\mathcal{A}_{y^*}^{\mathsf{WE}}$ as described above, but using y^* as the embedded y value. Note that as $\mathcal{B}_{\mathsf{OW}}$'s challenge is constructed as $y^* = H(esk^*)$ for a randomly chosen esk^*, the construction of $\mathcal{A}_{y^*}^{\mathsf{WE}}$ is identical to the above description. $\mathcal{B}_{\mathsf{OW}}$

then runs $\mathcal{E}$ for $\mathcal{A}^{\mathsf{WE}}_{y^*}$ to obtain a witness w, and forwards $w.esk$ as the solution in the onewayness game. Since HONEST occurs, $\mathcal{E}$ outputs a witness w corresponding to the honestly created record r^* for the challenge ciphertext c^* i.e. r^* must have been posted by the $\mathsf{HonestDec}$ or the Leak oracle. Furthermore, it must hold that $H(w.esk) = r^*.\sigma$, and due to the construction of $\mathcal{A}^{\mathsf{WE}}_{y^*}$, $r^*.\sigma = y^*$. Thus the obtained value $w.esk$ satisfies $w.esk = H^{-1}(y^*)$, and $\mathcal{B}_{\mathsf{OW}}$ therefore successfully wins the onewayness game. (Claim 2) □

Claim 3 *If* $\Pr[R_{\mathsf{FSPKE}}(x, w)|\neg\mathsf{HONEST}]$ *is non-negligible, there exists an adversary* $\mathcal{B}_{\mathsf{BLC}}$ *breaking the* $(\alpha, \beta, \ell_1, \ell_2)$*-distinguishable forking property of the blockchain with non-negligible advantage.*

Proof (Claim 3). The construction of $\mathcal{B}_{\mathsf{BLC}}$ is straightforward: $\mathcal{B}_{\mathsf{BLC}}$ simply runs $\mathcal{E}$, and returns its output $\mathbf{B}'$. Note, however, that $\mathcal{B}_{\mathsf{BLC}}$ plays the role of an adversary against the distinguishable forking property of the blockchain, and therefore must abide by the rules for this type of adversary. In particular, $\mathcal{B}_{\mathsf{BLC}}$ cannot control the honest parties participating in the blockchain protocol. Nevertheless, the simulation remains straightforward: $\mathcal{B}_{\mathsf{BLC}}$ simply corrupts the parties required by the underlying adversary $\mathcal{A}$, who will have a total stake fraction at most α, and forwards any messages to honest parties over the blockchain network as dictated by $\mathcal{A}$.

Since HONEST is assumed not to occur, the first valid record r' in $\mathbf{B}'$ allowing decryption of c^* does not correspond to the honestly generated record r^* in a Dec or Leak upon submission of c^* (recall that $\mathbf{B}'$ from a valid witness is required to contain a valid record allowing decryption of c^*). Furthermore, since FORGE is also assumed not to occur, r' cannot occur before r^* in the honest blockchain $\mathbf{B}'' \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st}_j)$ held by the honest user j. This implies that from the block in $\mathbf{B}'$ in which r' occurs, $\mathbf{B}'$ cannot be a prefix of $\mathbf{B}''$. Additionally, witness correctness implies that there are at least $\ell' = \ell_1 + \ell_2$ blocks after the block in which r' occurs, and that the last $\ell' - \ell_1$ blocks of these contain a combined stake fraction more than β. Hence, $\mathbf{B}'$ contradicts the $(\alpha, \beta, \ell_1, \ell_2)$-distinguishable forking property, which requires these blocks to contain a stake fraction less than $\alpha < \beta$. (Claim 3) □

Combining the above observations, we conclude that the existence of an adversary $\mathcal{A}$ with non-negligible advantage implies $\Pr[R_{\mathsf{FSPKE}}(x, w)]$ being non-negligible, which in turn implies that either the onewayness of H or the distinguishable forking property of BLC_V can be broken with non-negligible advantage due to (2) in combination with Claim 2 and Claim 3. This contradicts the assumption that H and BLC_V are secure, and we hence conclude that all $\mathcal{A}$ must have negligible advantage. Hence, Lemma 2 follows. (Lemma 2) □

Putting Lemma 1 and Lemma 2 together, we obtain:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{fs\text{-}IND\text{-}CPA}}_{\mathcal{A},\mathsf{FSPKE}}(\lambda) &= \left|\Pr[S] - \frac{1}{2}\right| \\
&= \left|\Pr[S|\mathsf{FORGE}]\Pr[\mathsf{FORGE}] + \Pr[S|\neg\mathsf{FORGE}]\Pr[\neg\mathsf{FORGE}] - \frac{1}{2}\right| \\
&\leq \left|\Pr[S|\neg\mathsf{FORGE}]\Pr[\neg\mathsf{FORGE}] - \frac{1}{2}\right| + \Pr[\mathsf{FORGE}] \\
&\leq \left|\Pr[S|\neg\mathsf{FORGE}](1 - \Pr[\mathsf{FORGE}]) - \frac{1}{2}\right| + \Pr\left[\mathsf{FORGE}\right] \\
&\leq neg(\lambda) + neg(\lambda) = neg(\lambda)
\end{aligned}$$

Hence, Theorem 2 follows. (Theorem 2) □

5 Discussion

Besides forward security, our construction provides several interesting properties which lead to advantages compared to existing approaches as well as additional security guarantees, but also impacts aspects such as decryption privacy. In the following, we discuss these in further detail.

Fixed Immutable Secret Keys. The unique feature of our construction is that forward security is achieved without key updates, and secret keys are short and immutable. This property provides several advantages.

Firstly, while the size of secret keys in most previous works [CHK03, GM15] depends on the number of key updates, our construction achieves a constant size secret key and furthermore does not impose a predetermined maximum number of possible key updates (such as Bloom filter encryption [DJSS18]).

Secondly, fixed immutable keys are interesting from an application point of view. For example, a fixed secret key can be embedded in secure read-only memory, which would provide an additional hardware-based defense against key compromise. Note that in our construction, the secret key is only required for signature generation, which is a standard functionality supported by most trusted platform modules (TPMs), and that the remaining part of decryption can be done without direct access to the secret key. In contrast, providing similar protection for a dynamically changing secret key of non-constant size is a harder task requiring a more advanced trusted execution environment, which in turn is more difficult and expensive to implement securely.

Lastly, a fixed secret key allows the key to be distributed among several independent devices or servers without introducing security concerns due to a potential lack of synchronization. Key distribution might be desirable e.g. if the same user uses several different devices or several servers are used to implement load balancing (here the servers look like one server from the outside). In this case, security concerns might arise for schemes implementing fine-grained forward security based on key update. For example, if a device decrypts a ciphertext

c, the local key of that device will be rendered useless for future decryptions of c to ensure forward security. However, unless the keys stored by all other devices are updated with respect to c, an adversary will still be able to decrypt c by compromising a device with a key that has not yet been updated. Hence, this creates a potentially significant synchronization problem. On the other hand, this problem is completely eliminated by a scheme with fixed secret keys, as there is no need to update keys to ensure security.

Decryption Privacy and Key Compromise Detection. Our construction requires the decryptor to post an appropriate message to the blockchain to decrypt a ciphertext. Specifically, Alice (holding the key pair pk_A and sk_A) is required to post a record r to the blockchain such that $\mathsf{Sig.Ver}(pk_A, r.\mathsf{id} \,\|\, r.\sigma, r.\mathsf{cert}) = 1$ holds to be able to decrypt a ciphertext $c = (\mathsf{id}, CT)$.

Note that as $r.\mathsf{cert}$ is publicly verifiable with respect to Alice's public key pk_A and id uniquely identifies c, anyone monitoring the blockchain, which is assumed to be publicly accessible, will be able to tell when Alice decrypts a specific ciphertext i.e. the construction does not provide Alice with privacy regarding decryption.

On the other hand, this gives the construction a unique security property not provided by existing schemes. More precisely, by monitoring the blockchain, Alice can detect if someone else is trying to decrypt a ciphertext using her private key. Hence, it is possible for Alice to detect a key compromise if the compromised key is ever attempted to be used for decryption. This property is not achievable if decryption can be done without any information being made public.

One-Time Decryption. In existing fine-grained encryption schemes without interaction [GHJL17, DJSS18, GM15], a ciphertext can be decrypted only once even by a legitimate user because an updated secret key cannot be used for decrypting past ciphertexts; the same limitation applies to our construction. Note that one-time decryption is an inherent property of fine-grained forward security.

Message Suppression Attacks and Mitigation. As we mentioned in Section 3.2, a standard fine-grained forward-secure scheme with perfect correctness inherently does not protect against message suppression attacks [BG20]. A message suppression attack is a man-in-the-middle attack where the attacker is assumed to control the communication between encryptor and decryptor and simply does not deliver a given ciphertext c. Then, if the attacker is allowed to compromise the secret key, he will be able to decrypt c due to the perfect correctness of the scheme and the fact that c has not been attempted to decrypt by the decryptor[11].

To mitigate the attack in our construction, we can introduce *decryption expiration* (similar to eventual forward security [BG20]) by checking in the witness

[11] In a scheme based on periodical key updates, this type of attack does not work assuming the key has been updated after the ciphertext was constructed and the attacker compromises this updated key. However, schemes based on periodical key updates only achieve coarse-grained forward security (eventual forward security [BG20]) and are still vulnerable to the attack until the key is updated.

relation $R_{\mathsf{FSPKE}}(x, w)$ that the number of blocks in $w.\mathbf{B}'$ extended from $x.\mathbf{B}$ is less than a predefined expiration threshold. This ensures that if the adversary does not compromise the secret key before the extension of the blockchain passes the threshold, he will not be able to decrypt the intercepted ciphertext. However, this will also require the legitimate decryptor to decrypt the ciphertext before the expiration, as he would otherwise lose the ability to do so. Finally note that this change does not interfere with the property that once the decryptor has decrypted a ciphertext, this can no longer be decrypted by an adversary compromising the decryption key i.e. fine-grained forward security is maintained.

Acknowledgments

This work was supported in part by JSPS KAKENHI Grant Number 20K11807.

References

AFH+20. Martin R. Albrecht, Pooya Farshim, Shuai Han, Dennis Hofheinz, Enrique Larraia, and Kenneth G. Paterson. Multilinear Maps from Obfuscation. *J. Cryptol.*, 33(3):1080–1113, 2020.

AGJ19. Nimrod Aviram, Kai Gellert, and Tibor Jager. Session Resumption Protocols and Efficient Forward Security for TLS 1.3 0-RTT. In *Advances in Cryptology – EUROCRYPT 2019*, pages 117–150, 2019.

BG20. Colin Boyd and Kai Gellert. A Modern View on Forward Security. In *The Computer Journal*, volume 64, pages 639–652, 2020.

BH15. Mihir Bellare and Viet Tung Hoang. Adaptive Witness Encryption and Asymmetric Password-Based Cryptography. In *Public-Key Cryptography – PKC 2015*, pages 308–331, 2015.

BIJ+20. James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine Determinant Programs: A Framework for Obfuscation and Witness Encryption. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020*, volume 151 of *LIPIcs*, pages 82:1–82:39, 2020.

BIOW20. Ohad Barta, Yuval Ishai, Rafail Ostrovsky, and David J. Wu. On Succinct Arguments and Witness Encryption from Groups. In *Advances in Cryptology - CRYPTO 2020*, volume 12170, pages 776–806, 2020.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

CCG16. K. Cohn-Gordon, C. Cremers, and L. Garratt. On Post-compromise Security. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 164–178, 2016.

CHK03. Ran Canetti, Shai Halevi, and Jonathan Katz. A Forward-Secure Public-Key Encryption Scheme. In *Advances in Cryptology — EUROCRYPT 2003*, pages 255–271, 2003.

DJSS18. David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom Filter Encryption and Applications to Efficient Forward-Secret 0-RTT Key Exchange. In *Advances in Cryptology – EUROCRYPT 2018* , pages 425–455, 2018.

DPS19. Phil Daian, Rafael Pass, and Elaine Shi. Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake. In *Financial Cryptography and Data Security*, pages 23–41, 2019.

GG17. Rishab Goyal and Vipul Goyal. Overcoming Cryptographic Impossibility Results Using Blockchains. In *Theory of Cryptography*, pages 529–561, 2017.

GGSW13. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness Encryption and Its Applications. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 467–476, 2013.

GHJL17. Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT Key Exchange with Full Forward Secrecy. In *Advances in Cryptology – EUROCRYPT 2017*, pages 519–548, 2017.

GKM+20. Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and Retrieving Secrets on a Blockchain. *IACR Cryptol. ePrint Arch.*, page 504, 2020.

GKP+13. Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to Run Turing Machines on Encrypted Data. In *Advances in Cryptology – CRYPTO 2013*, pages 536–553, 2013.

GM15. Matthew D. Green and Ian Miers. Forward Secure Asynchronous Messaging from Puncturable Encryption. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, page 305–320, 2015.

Gro16. Jens Groth. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology - EUROCRYPT 2016*, volume 9666, pages 305–326, 2016.

JLS21. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability Obfuscation from Well-Founded Assumptions. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 60–73, 2021.

LJKW18. Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to Build Time-Lock Encryption. *Des. Codes Cryptography*, 86(11):2549–2586, 2018.

NSN21. Seiya Nuta, Jacob C. N. Schuldt, and Takashi Nishide. Forward-secure public key encryption without key update from proof-of-stake blockchain. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology – INDOCRYPT 2021*, pages 436–461, Cham, 2021. Springer International Publishing.

PS14. David Pointcheval and Olivier Sanders. Forward Secure Non-Interactive Key Exchange. In *Security and Cryptography for Networks*, pages 21–39, 2014.

Res. Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC8446.