Fair Signature Exchange

Hossein Hafezi[2], Aditi Partap[4], Sourav Das[3], and Joseph Bonneau[1,2]

[1]A16Z Crypto Research
[2]New York University (NYU)
[3]University of Illinois Urbana-Champaign (UIUC)
[4]Stanford University
h.hafezi@nyu.edu, aditi712@stanford.edu,souravd2@illinois.edu, jcb@cs.nyu.edu

Abstract. We introduce the concept of Fair Signature Exchange (FSE). FSE enables a client to obtain signatures on multiple messages in a *fair* manner: the client receives all signatures if and only if the signer receives an agreed-upon payment. We formalize security definitions for FSE and present a practical construction based on the Schnorr signature scheme, avoiding computationally expensive cryptographic primitives such as SNARKs. Our scheme imposes minimal overhead on the Schnorr signer and verifier, leaving the signature verification process unchanged from standard Schnorr signatures. Fairness is enforced using a blockchain as a trusted third party, while exchanging only a constant amount of information on-chain regardless of the number of signatures exchanged. We demonstrate how to construct a batch adaptor signature scheme using FSE, and our FSE construction based on Schnorr results in an efficient implementation of a batch Schnorr adaptor signature scheme for the discrete logarithm problem. We implemented our scheme to show that it has negligible overhead compared to standard Schnorr signatures. For instance, exchanging 2^{10} signatures on the Vesta curve takes approximately 80ms for the signer and 300ms for the verifier, with almost no overhead for the signer and 2x overhead for the verifier compared to the original Schnorr protocol. Additionally, we propose an extension to *blind* signature exchange, where the signer does not learn the messages being signed. This is achieved through a natural adaptation of blinded Schnorr signatures.

1 Introduction

Consider a scenario where a server holds a signing key and a client wishes to obtain signatures on a batch of messages in exchange for payment. For example, the client might be purchasing a set of certificates, or a set of one-time use tokens. In most existing applications, clients are forced to trust the server and pay upfront, receiving the signatures later. However, this model is unsuitable for decentralized environments where no party can be inherently trusted. Conversely, the server cannot send the signatures upfront, as the client's reliability to pay cannot be guaranteed.

In this paper, we introduce a new cryptographic primitive called *Fair Signature Exchange* (FSE). The primary goal of FSE is to enable a signer to *atomically* and *conditionally* exchange signatures on a batch of messages. Exchanging signatures is a special case of *fair exchange*, and it is well-established that this is impossible for two parties to achieve without the help of a trusted third party (TTP) [21]. A natural idea is to instantiate the TTP using a blockchain. Tas et al. [25] proposed doing so for fair exchange of data as follows: the server first transmits encrypted data to the client, along with a commitment to the decryption key and a proof demonstrating that the commitment corresponds to the decryption key of the ciphertext. Subsequently, the client posts a conditional payment transaction on the blockchain, which is activated only if the commitment to the decryption key is correctly opened on the blockchain. This method is sound and requires only constant communication on the blockchain. However, it suffers from several drawbacks: the server-to-client communication overhead is high due to encryption overhead, and server-side computations are costly because the server must generate a Succinct Non-interactive Argument of Knowledge (SNARK) proof that the commitment opens to a valid decryption key corresponding to the encrypted message.

In this work, we propose a simpler primitive called *Fair Signature Exchange* (FSE). FSE provides a flexible framework involving two parties, signer, and client, both of which have access to a TTP. The client possesses a set of messages $\{m_i\}$ and a public key pk, while the signer holds the corresponding secret key sk. The objective is for client to conditionally obtain signatures $\{\sigma_i\}$ on messages $\{m_i\}$ from signer, in exchange for some agreed-upon payment. We present am FSE construction which minimizes communication using the blockchain while also reducing server-to-client communication. Importantly, our scheme does not rely on arguments of knowledge or SNARKs, significantly reducing the computational burden on the server. Our construction is based on the Schnorr signature scheme. We provide formal definitions for FSE akin to standard signature security definitions and prove the security of our construction. Additionally, we implement our scheme and show that it has almost zero overhead on the signer and only 2x overhead on the verifier compared to the original Schnorr protocol and importantly, the verification procedure remains the same. FSE can used to build the batch version of *adaptor signatures* for the discrete logarithm relation, which facilitates the atomic exchange of a single signature and a discrete logarithm of a public group element. An adaptor signature is a cryptographic primitive that efficiently facilitates such atomic exchange using a blockchain as a trusted third party (we elaborate more on adaptor signatures in Section 2). Adaptor signatures are widely used in decentralized finance (DeFi), payment channels (batch atomic swaps) and multi-chain bridges; however, unlike FSE, they are limited to exchanging one signature (or payment) at a time, which becomes impractical when dealing with multiple accounts or transactions simultaneously.

2 Related work

The closest to our work are the fair-data exchange (FDE) construction [25], and adaptor signatures [15]. Our work is inspired by FDE but for signatures. We can exchange data significantly more efficiently by focusing on exchanging signatures.

Fair data exchange (FDE). In the FDE setting, the client holds a commitment to some data, which the server possesses. The concept of fairness is defined in two parts: the client cannot obtain the data without paying the server (*Server Fairness*) and the server cannot receive payment without revealing the data to the client (*Client Fairness*). Tas et al., in [25] presents two concrete constructions of FDE. In both constructions, the server encrypts the data with a random key. It sends the ciphertexts to the client, along with a commitment to the key, and a zero-knowledge proof that the ciphertexts encrypt the correct data with the key and commitment to the key opens to the key. The server and client then exchange the key for some agreed-upon payment via the trusted third party. However, their constructions are expensive since they require data encryption and arguments of knowledge.

Akin to FDE, in FSE, signer has some secret key sk and client has the corresponding public input pk, but this secret value sk is never revealed to the client. Furthermore, unlike the setting in [25], FSE must address the risk of a faulty client attempting to learn a signature on one message by querying signatures for other messages. This was not a concern in the server fairness definition of [25]. Additionally, in our scheme, we only rely on the hardness of discrete logarithm, without requiring expensive SNARKs, as in [25].

Adaptor signatures. An adaptor signature is a cryptographic protocol that extends standard digital signatures to enable conditional commitments. In this scheme, the signer generates an incomplete signature with respect to a NP statement Y. Any client with the corresponding witness y can obtain the full signature. Adaptor signatures are extractable, meaning that the signer can compute the witness y using the full signature and the corresponding incomplete signature. This mechanism is widely used in trustless atomic swaps within DeFi [16] and payment channels [10, 22, 19]. Adaptor signatures enable trustless cross-chain exchanges by embedding a cryptographic secret into partially signed transactions, ensuring atomicity. Consider Alice (owning Bitcoin) and Bob (owning Ethereum) who agree to exchange assets using a shared secret x and its hash $H(x)$. Alice locks 1 BTC in a Bitcoin hash time-locked contract (HTLC), allowing Bob to claim it only by revealing x within a deadline. Instead of fully signing the transaction, Alice provides an adaptor signature, which can only be finalized by revealing x. Concurrently, Bob locks 10 ETH in an Ethereum HTLC, allowing Alice to claim it with x. This setup ensures the exchange cannot be completed unless both parties act. Bob finalizes Alice's adaptor signature to claim the Bitcoin, which reveals x in the process. Alice then uses x to unlock the Ethereum on Bob's HTLC. If either party fails to act before the deadlines, the locked funds are returned to their respective owners, preserving security. This mechanism ensures atomicity: either both exchanges are completed, or neither happens. One attractive feature of adaptor signatures is that they do not require the blockchain to support smart contracts, only require the support of HTLC. As we mentioned earlier, FSE can be used to extend adaptor signatures for the discrete logarithm relation to the batch setting, wherein the client can get multiple signatures in exchange for a single secret discrete logarithm value. Batching is particularly useful for complex DeFi scenarios e.g. batch atomic swaps. The key efficiency of our scheme lies in using a single secret value to complete all transactions (signatures), regardless of the batch size. The signer can extract the secret witness when any one of the signatures from the batch is posted onchain by the client. In contrast, classic adaptor signature schemes would require a unique secret for each transaction, resulting in a linear increase in blockchain communications.

Functional adaptor signatures. Functional Adaptor Signatures (FAS) [26] is an extension of adaptor signatures that enables a server and a client to fairly exchange a signature for a function evaluation $f(x)$ where x is some secret value. In this protocol, the server learns nothing about x beyond the value of $f(x)$, and can compute $f(x)$ if and only if the client gets a valid signature. On a similar vein as FDE, the client encrypts x using functional encryption to get c_x. The client and server then use adaptor signatures to exchange the functional secret key for f for a signature. The server can then decrypt c_x using the functional secret key to obtain $f(x)$.

3 Preliminaries

Notations. For any integer n, we use $[n]$ to denote the ordered set $\{1, 2, \ldots, n\}$. For any non-empty set S, we let $s \leftarrow\$ S$ indicate that s is sampled uniformly at random from S, and we use $|S|$ to denote the size of S. We use λ to denote the security parameter. We denote negligible functions with $\mathsf{negl}(\lambda)$. A machine is called probabilistic polynomial time (PPT) if it is a probabilistic algorithm that runs in $\mathsf{poly}(\lambda)$ time. All algorithms are probabilistic unless stated otherwise. By $y \leftarrow A(x_1, \ldots, x_n)$, we denote the operation of running algorithm A on inputs $(x_1, \ldots, x_n)$ with uniformly random coins and letting y denote the output. If A has oracle access to some algorithm E, we write $y \leftarrow A^{\mathsf{E}(\cdot)}(x_1, \ldots, x_n)$.

Computational assumptions. Let GGen be a group generation algorithm that on input 1^λ outputs the description of a prime order group $\mathbb{G}$. The description contains the prime order p, a generator $g \in \mathbb{G}$, and a description of the group operation. We will use the multiplicative notation for the group operation. Our protocol assumes the standard discrete logarithm (DL) assumption in group $\mathbb{G}$, which we formally define in Definition 6. We denote a random oracle using H that we assume is randomly sampled from random oracle space $\mathcal{H}$.

Digital signature. It is a cryptographic protocol consisting of four algorithms $\Pi = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, defined as follows:

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm takes as input a security parameter λ and outputs system parameters pp, e.g. including the description of the message space and the key space.
- $\mathsf{Gen}(\mathsf{pp})$: The key generation algorithm takes the system parameters pp and outputs key pair $(\mathsf{sk}, \mathsf{pk})$, where sk is the secret (signing) key and pk is the public (verification) key.
- $\mathsf{Sign}(\mathsf{sk}, m)$: On input secret key sk and a message m, outputs a signature σ.
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$: The verification algorithm is a deterministic algorithm that takes the public key pk, a message m, and a signature σ, and outputs a bit $b \in \{0, 1\}$. The output $b = 1$ indicates that the signature is valid, and $b = 0$ indicates that it is invalid.

A digital signature scheme is secure if it satisfies correctness and *strong* unforgeability as defined below:

Definition 1 (Siganture Correctness). *Signature scheme* $\Pi = (\mathsf{Setup}, \mathit{Gen}, \mathsf{Sign}, \mathit{Verify})$ *is correct, if the following holds:*

$$\Pr\left[b = 1 \;\middle|\; \begin{array}{c} \mathit{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda) \\ (\mathsf{sk},\ \mathsf{pk}) \leftarrow \Pi.\mathit{Gen}(\mathit{pp}) \\ \sigma \leftarrow \Pi.\mathsf{Sign}(\mathsf{sk},\ m) \\ b \leftarrow \Pi.\mathit{Verify}(\mathsf{pk},\ m,\ \sigma) \end{array} \right] = 1.$$

Definition 2 (Signature Strong Unforgeability). *A signature scheme* $\Pi = (\mathsf{Setup}, \mathit{Gen}, \mathsf{Sign}, \mathit{Verify})$ *has* strong existential unforgeability under chosen-message attack*, if any* PPT *adversary* $\mathcal{A}$ *wins the game defined in Figure 1 with negligible probability.*

```
Unforgeability_A^Π(1^λ)
1 : $\mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda)$
2 : $(\mathsf{pk},\ \mathsf{sk}) \leftarrow \Pi.\mathsf{KGen}(\mathsf{pp});\ \mathcal{Q} := \phi$
3 : $(m^*,\ \sigma^*) \leftarrow \mathcal{A}^{\mathsf{SIGN}(\cdot)}(\mathsf{pp},\ \mathsf{pk})$
4 : return $(m^*, \sigma^*) \notin \mathcal{Q} \wedge \Pi.\mathsf{Verify}(\mathsf{pk},\ m^*,\ \sigma^*) = 1$
```

```
Oracle SIGN(m)
1 : $\sigma \leftarrow \Pi.\mathsf{Sign}(\mathsf{sk},\ m)$
2 : $\mathcal{Q} = \mathcal{Q} \cup \{(m, \sigma)\}$
3 : return $\sigma$
```

Fig. 1: Unforgeability game for signature scheme Π

The Schnorr signature scheme [24]. Let $(\mathbb{G},\ p,\ g) \leftarrow \mathsf{GGen}(1^\lambda)$. Let $\mathsf{H}_{\mathsf{Sig}} : \mathbb{G}^2 \times \mathcal{M} \rightarrow \mathbb{Z}_p$ be a hash function modelled as a random oracle, where $\mathcal{M}$ is the message space. The signing key $\mathsf{sk} \in \mathbb{Z}_p$ is a random field element, and $\mathsf{pk} := g^{\mathsf{sk}} \in \mathbb{G}$ is the corresponding public verification key. The signature σ on a message m is then $(R, s) \in \mathbb{G} \times \mathbb{Z}_p$. To validate a signature $\sigma = (R, s)$ on a message m, a validator first computes $c := \mathsf{H}_{\mathsf{Sig}}(R, \mathsf{pk}, m)$ and checks that $g^s = R \cdot \mathsf{pk}^c$. Schnorr signatures have been proven strongly unforgeable in the random oracle model [24, 3].

4 Fair Signature Exchange

4.1 Definitions

A *Fair Signature Exchange* (FSE) protocol is a protocol between client and signer involving a transparent payment environment E as a trusted third party (TTP) that holds money under addresses belonging to the other parties. The TTP can transfer money from one party's address to another but requires a transaction authorizing the transfer with the sender's signature. It is transparent in the sense that any message sent to E eventually becomes visible to all other parties. The FSE protocol $\mathsf{FSE}[\Pi]$ initialized with a signature scheme Π, consists of PPT algorithms (Setup, KeyGen, PSign, $\mathsf{PVerify}$, $\mathsf{Execute}$, $\mathsf{Recover}$). Both the client and the signer parties are assumed to have access to the functionalities of Π. For the sake of simplicity from now on, we write FSE instead of $\mathsf{FSE}[\Pi]$.

– $\mathsf{FSE.Setup}(1^\lambda) \rightarrow \mathsf{pp}$ is a randomized algorithm that outputs the public parameters for the system (e.g., the description of appropriate spaces). All the following algorithms and protocols implicitly take the pp as input.
– $\mathsf{FSE.KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{pk}, \mathsf{sk})$ is a randomized algorithm that samples a secret signing key sk and computes the corresponding public key pk.
– $\mathsf{FSE.PSign}(\mathsf{sk},\ \{\mathsf{m}_i\}_{i\in[n]}) \rightarrow (\{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ \mathsf{com}_k)$ is a randomized algorithm that takes as input the secret key sk and a batch of messages $\{m_i\}_{i\in[n]} \in \mathcal{M}^n$. It samples an encryption key $k \leftarrow\!\!\$\ \mathcal{K}$, computes partial (masked) signatures $\{\tilde{\sigma}_i\}_{i\in[n]}$ along with auxiliary data aux and a commitment com_k to the key. Here $\mathcal{M}$ denote the message space of Π and $\mathcal{K}$ is the key space. We also use $\mathsf{FSE.PSign}(\mathsf{sk},\ \{\mathsf{m}_i\}_{i\in[n]};\ k)$ to denote the above process.
– $\mathsf{FSE.PVerify}(\mathsf{pk},\ \{\mathsf{m}_i\}_{i\in[n]},\ \{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ \mathsf{com}_k) \rightarrow \{0,1\}$ is a deterministic algorithm that on input public key pk, partial signatures $\{\tilde{\sigma}_i\}_{i\in[n]}$ auxiliary data aux and a key commitment com_k, checks whether the partial signatures are valid.
– $\mathsf{FSE.Execute}(\langle\mathsf{client}(\mathsf{com}_k,\ \mathsf{tok}) \leftrightarrow \mathsf{E} \leftrightarrow \mathsf{signer}(k)\rangle)$. client and signer communicate independently with the TTP environment E, at the end of which client receives the opening k to commitment com_k, and signer receives some token tok. This interaction can be realized with the following algorithms, wherein $\mathsf{FSE}^{(\mathsf{E})}$ indicates the algorithm having access to the environment E:

$$\begin{aligned} \mathsf{st}_{\mathsf{E},0} &\leftarrow \mathsf{FSE}^{(\mathsf{E})}.\mathsf{Init}() \\ \mathsf{st}_{\mathsf{E},1} &\leftarrow \mathsf{FSE}^{(\mathsf{E})}.\mathsf{ExeClient}_0(\mathsf{com}_k, \mathsf{tok}) \\ (\mathsf{tok}_s, \mathsf{st}_{\mathsf{E},2}) &\leftarrow \mathsf{FSE}^{(\mathsf{E})}.\mathsf{ExeSigner}_0(k) \end{aligned}$$

In essence, the interaction on E is successful if the signer receives the payment, i.e. $\mathsf{tok}_s \neq \perp$.
– $\mathsf{FSE.Recover}(\{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ k) \rightarrow \{\sigma_i\}_{i\in[n]}$ is a deterministic algorithm that, on input partial signatures $\{\tilde{\sigma}_i\}_{i\in[n]}$, auxiliary data aux and the decryption key k, outputs signatures $\{\sigma_i\}_{i\in[n]}$.

We require the FSE protocol to meet the following properties: *completeness*, *signer fairness*, *client fairness*. Completeness ensures that if both parties honestly follow the protocol, then in the end, the signer receives the tokens, and the client receives signatures. The signer fairness property ensures that a malicious client cannot obtain more valid signatures than it has paid for. Lastly, the client fairness property guarantees that a dishonest signer would not be paid without committing to valid signatures and subsequently revealing them to the client. We formalize these properties as follows.

Definition 3 (Completeness). *We say that a fair signature exchange scheme* $\mathsf{FSE}[\Pi]$ *is complete if for any polynomial batch size* $n = n(\lambda)$*, and any set of polynomially long messages* $\{m_i\}_{i\in[n]} \in \mathcal{M}^n$*, the following holds:*

$$\Pr\left[\begin{array}{c|c} & \mathsf{pp} \leftarrow \mathsf{FSE.Setup}(1^\lambda) \\ & (\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{FSE.KeyGen}();\ k \leftarrow\!\!\$\ \mathcal{K} \\ & (\{\tilde{\sigma}_i\}_{i\in[n]}, \mathsf{aux}, \mathsf{com}_k) \leftarrow \mathsf{FSE.PSign}(\mathsf{sk}, \{m_i\}_{i\in[n]};\ k) \\ b = 1\ \wedge\ \mathsf{tok}_s \neq \perp\ \wedge & b := \mathsf{FSE.PVerify}(\mathsf{pk}, \{\tilde{\sigma}_i\}_{i\in[n]}, \mathsf{aux}, \mathsf{com}_k) \\ \forall i \in [n] : \Pi.\mathsf{Verify}(\mathsf{pk},\ m_i,\ \sigma_i) = 1 & \{\sigma_i\}_{i\in[n]} := \mathsf{FSE.Recover}(\{\tilde{\sigma}_i\}_{i\in[n]}, \mathsf{aux}, k) \\ & \mathsf{st}_{\mathsf{E},0} \leftarrow \mathsf{FSE}^{(\mathsf{E})}.\mathsf{Init}() \\ & \mathsf{st}_{\mathsf{E},1} \leftarrow \mathsf{FSE}^{(\mathsf{E})}.\mathsf{ExeClient}_0(\mathsf{com}_k, \mathsf{tok}) \\ & (\mathsf{tok}_s, \mathsf{st}_{\mathsf{E},2}) \leftarrow \mathsf{FSE}^{(\mathsf{E})}.\mathsf{ExeSigner}_0(k) \end{array}\right] = 1$$

```
Game $\mathsf{SignerFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda)$
1 : $\mathsf{pp} \leftarrow \mathsf{FSE.Setup}(1^\lambda)$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FSE.KGen}(\mathsf{pp}, 1^\lambda)$
3 : $\mathcal{S} := \phi, \mathcal{K} := \phi, \mathcal{N} := \phi, c = 0, t = 0$
4 : $\{(m_i^*, \sigma_i^*)\}_{i \in t'} \leftarrow \mathcal{A}^{\mathsf{SIGN}(\cdot), \mathsf{EXEC}(\cdot)}(\mathsf{pp}, \mathsf{pk})$
5 : $\textbf{return } t < t' \wedge$
6 :     $\forall i \in [t'] : \Pi.\mathsf{Verify}(\mathsf{pk}, m_i^*, \sigma_i^*) = 1 \wedge$
7 :     $\forall i \neq j \in [t'] : m_i^* \neq m_j^*$

Oracle $\mathsf{SIGN}(\{m_i\}_{i \in [n]})$
1 : $c \leftarrow c + 1$
2 : $k \leftarrow\$ \mathcal{K}; \mathcal{K}[c] := k$
3 : $msg \leftarrow \mathsf{FSE.PSign}(\mathsf{sk}, \{m_i\}_{i \in [n]}; k)$
4 : $\mathcal{S} := \mathcal{S} \cup \{c\}; \mathcal{N}[c] = n$
5 : $\textbf{return } (c, msg)$

Oracle $\mathsf{EXEC}(j, \mathsf{com}_k)$
1 : $\textbf{if } j \notin \mathcal{S} \vee \mathsf{com}_k \neq com(\mathcal{K}[j]) \textbf{ return } \perp$
2 : $k_j := \mathcal{K}[j]; n_j := \mathcal{N}[j]$
3 : $t = t + n_j$
4 : $\textbf{return } k_j$
```

Fig. 2: Signer fairness game $\mathsf{SignerFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda)$ for a fair signature exchange protocol $\mathsf{FSE}[\Pi]$. Note that we consider a strong adversary that can start multiple signing sessions, and additionally, run Execute multiple times for each session.

Definition 4 (Signer Fairness). *We say the scheme* FSE *has signer-fairness if for all* PPT *adversaries* $\mathcal{A}$*, the following quantity is negligible in* λ*:*

$$\mathsf{Adv}_{\mathcal{A},\mathsf{FSE}[\Pi]}^{\mathrm{sf}}(\lambda) = \Pr[\mathsf{SignerFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda) = 1]$$

where the signer fairness game is as defined in Figure 2.

Definition 5 (Client Fairness). *We say the scheme* FSE *has client-fairness if any* PPT *adversary* $\mathcal{A}$ *wins the client fairness game in Figure 3 game with negligible probability.*

```
Game $\mathsf{ClientFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda)$
1 : $\mathsf{pp} \leftarrow \mathsf{FSE.Setup}(1^\lambda)$
2 : $(\mathsf{pk}, st) \leftarrow \mathcal{A}(\mathsf{pp})$
3 : $\mathcal{B} := \phi, \mathcal{I} := \phi, \mathcal{F} := \phi, \mathcal{S} := \phi, c \leftarrow 0$
4 : $(st) \leftarrow \mathcal{A}^{\mathsf{VERIFY}(\cdot), \mathsf{EXEC}(\cdot)}(st)$
5 : $\textbf{if } \exists\, i \in [c] \text{ s.t. } \mathcal{B}[i] = 1 \wedge$
6 :   $\exists\, (m, \sigma) \in \mathcal{F}[i] : \Pi.\mathsf{Verify}(\mathsf{pk}, m, \sigma) = 0 :$
7 :   $\textbf{return } 1$
8 : $\textbf{else return } 0$

Oracle $\mathsf{VERIFY}(\{m_i\}_{i \in [n]}, (\{\tilde{\sigma}_i\}, \mathsf{aux}, \mathsf{com}_k))$
1 : $c \leftarrow c + 1$
2 : $\mathcal{B}[c] \leftarrow \mathsf{FSE.PVerify}(\mathsf{pk}, \{\tilde{\sigma}_i\}_{i \in [n]}, \mathsf{aux}, \mathsf{com}_k)$
3 : $\mathcal{I}[c] \leftarrow (\{m_i\}_{i \in [n]}, \{\tilde{\sigma}_i\}_{i \in [n]}, \mathsf{aux}, \mathsf{com}_k)$
4 : $\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$

Oracle $\mathsf{EXEC}(j, k, \mathsf{tok})$
1 : $\textbf{if } j \notin \mathcal{S} \textbf{ return } \perp$
2 : $(\{m_i\}_{i \in [n]}, \{\tilde{\sigma}_i\}_{i \in [n]}, \mathsf{aux}, \mathsf{com}_k) := \mathcal{I}[j]$
3 : $\mathsf{st}_{\mathsf{E},0} \leftarrow \mathsf{FSE.Init}()$
4 : $\mathsf{st}_{\mathsf{E},1} \leftarrow \mathsf{FSE.ExeClient}_0(\mathsf{com}_k, \mathsf{tok})$
5 : $(\mathsf{tok}_s, \mathsf{st}_{\mathsf{E},2}) \leftarrow \mathsf{FSE.ExeSigner}_0(k)$
6 : $\{\sigma_i\}_{i \in [n]} \leftarrow \mathsf{FSE.Recover}(\{\tilde{\sigma}_i\}_{i \in [n]}, \mathsf{aux}, k)$
7 : $\mathcal{S} \leftarrow \mathcal{S} \setminus \{j\}$
8 : $\textbf{if } \mathsf{tok}_s = \perp \textbf{ return } \perp$
9 : $\mathcal{F}[j] \leftarrow \{(m_i, \sigma_i)\}_{i \in [n]}$
10 : $\textbf{return } \{\sigma_i\}_{i \in [n]}$
```

Fig. 3: Client fairness game $\mathsf{ClientFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda)$ for a fair signature exchange protocol FSE.

5 Design

We now describe our construction of the FSE protocol. For simplicity, we assume client has a single message m and seeks a signature from signer. We consider the case of a vector of messages later in the section.

1. client needs a proof that the commitment com actually opens up to the correct value s, that is, that $\sigma = (R,\, s)$ is a valid signature for the message of client. This is important so that client does not have to trust signer blindly.
2. The final step, wherein signer reveals the commitment opening s on E and E subsequently transfers the payment to signer, will be costly when running the protocol for a vector of messages because the communication on E will grow linearly with the number of messages to be signed. This is not ideal, since communication on E is much more expensive than direct communication between signer and client. Ideally, we would like the communication to remains constant and independent of the number of messages.

```
$\mathsf{FSE.Setup}(1^\lambda)$
1 :  $(\mathbb{G},\, p,\, g) \leftarrow\$\, \mathsf{GGen}(1^\lambda)$
2 :  $\mathsf{H} \leftarrow\$\, \mathcal{H}$
3 :  return $\mathsf{pp} := (\mathbb{G},\, p,\, g,\, \mathsf{H})$

$\mathsf{FSE.KeyGen}()$
1 :  $\mathsf{sk} \leftarrow\$\, \mathbb{Z}_p$
2 :  return $(\mathsf{sk},\, \mathsf{pk} := g^x)$

$\mathsf{FSE.PVerify}(\mathsf{pk}, \{m_i\}_{i\in[n]},\, \{\tilde{\sigma}_i\}_{i\in[n]},\, \mathsf{aux},\, \mathsf{com}_k)$
1 :  Parse $\{R_i\}_{i\in[n]} \leftarrow \mathsf{aux}$
2 :  $\forall\, i \in [n] : c_i := \mathsf{H}(R_i,\, m_i)$
3 :  return true if $\forall\, i \in [n]$ :
4 :      $\mathsf{com}_i := R_i \cdot \mathsf{pk}^{c_i}$
5 :      $g^{2\tilde{\sigma}_i} \stackrel{?}{=} \mathsf{com}_k \cdot \mathsf{com}_i$

$\mathsf{FSE.PSign}(\mathsf{sk}, \{m_i\}_{i\in[n]})$
1 :  $k \leftarrow\$\, \mathbb{Z}_p$; $\mathsf{com}_k := g^k$
2 :  $\forall i \in [n]$ :
3 :      $r_i \leftarrow\$\, \mathbb{Z}_p$; $R_i := g^{r_i}$
4 :      $c_i := \mathsf{H}(R_i,\, m_i)$
5 :      $s_i := r_i + c_i \cdot \mathsf{sk}$
6 :      $\tilde{\sigma}_i := \frac{k + s_i}{2}$
7 :  return $(\{\tilde{\sigma}_i\}_{i\in[n]},\, \mathsf{aux} := \{R_i\}_{i\in[n]},\, \mathsf{com}_k)$

$\mathsf{FSE.Recover}(\{\tilde{\sigma}_i\}_{i\in[n]},\, \mathsf{aux},\, k)$
1 :  Parse $\{R_i\}_{i\in[n]} \leftarrow \mathsf{aux}$
2 :  $\forall i \in [n] : s_i := 2 \cdot \tilde{\sigma}_i - k$
3 :  return $\{\sigma_i := (R_i,\, s_i)\}_{i\in[n]}$
```

Fig. 4: FSE protocol for fair signature exchange of Schnorr signatures

We address the first issue by leveraging the inherent structure of Schnorr signatures. Specifically, recall that the Schnorr verification equation is $g^s \stackrel{?}{=} R \cdot \mathsf{pk}^c$ where $c = \mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m)$. Hence, the signer can simply send R and g^s as a commitment to s. client can easily verify this commitment by just checking the Schnorr verification equation. Importantly, signer does not need to send any additional data to validate these commitments. This eliminates the need for an argument of knowledge, making our protocol significantly more efficient than a general-purpose Fair Data Exchange (FDE). To address the second issue, consider the scenario in which parties need to exchange signatures on n messages. In a naive approach, client and signer would send the commitments $\{g^{s_i}\}$ and corresponding openings $\{s_i\}$ on E, resulting in linear communication in n. Instead, we propose a modified signing protocol, wherein the signer samples a symmetric encryption key k, and sends encrypted values $\{\tilde{\sigma}_i\}$ derived from signatures $\{s_i\}$ along with a commitment to the key k (Figure 5). More formally, signer computes the partial signatures $\tilde{\sigma}_i = (k + s_i)/2$ as illustrated in Figure 4. Later we prove $\tilde{\sigma}_i$ effectively *hides* the underlying signature s_i. signer sends these along with a binding commitment to the key, $\mathsf{com}_k = g^k$, to client. To exchange a batch of n signatures, client and signer only need to transmit the decryption key k and its commitment com_k through E, reducing the communication complexity in E from linear in the batch size to constant. Once client obtains k, they can easily recover the original signatures s_i from the masked values $\tilde{\sigma_i}$. Figure 5 depicts the masking process – since the client only sees the partial signatures, we can prove that the client cannot compute the signatures without the decryption key k via E, based on the discrete log assumption.

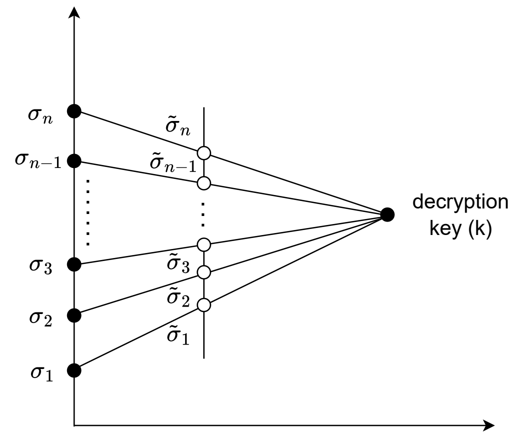


Fig. 5: Let σ_i represent the field elements of the Schnorr signatures requested by the client. k denotes the symmetric encryption key, which is uniformly randomly sampled by the server. The expression $\tilde{\sigma}_i = \frac{\sigma_i + k}{2}$ represents the "encryption" of σ_i under the key k. The client and server now only need to exchange k via the transparent environment. Crucially, we can securely use the same key k for all n signatures since the signature values σ_i are random and independent, from the view of the client.

An overview of the whole interaction between parties signer, client and E can be seen in Figure 6. Figure 4 provides a formal overview of our proposed scheme. Additionally, Figure 7 presents the interaction via E to exchange the key k for tokens.

5.1 Analysis

Theorem 1. *The fair signature exchange scheme* FSE *described in Figure 4 is complete.*

Proof. Let $\mathsf{pp} = (\mathbb{G}, p, g, \mathsf{H})$ be the parameters output by the setup algorithm $\mathsf{FSE.Setup}(1^\lambda)$ and $\{m_i\}_{i\in[n]} \in \mathcal{M}^n$ be any set of messages that $n \in \mathrm{poly}(\lambda)$. Suppose $(\mathsf{sk}, \mathsf{pk})$ is the key pair output by $\mathsf{FSE.Gen}()$, where $\mathsf{pk} = g^{\mathsf{sk}}$. Let $k \leftarrow \mathbb{Z}_p$, and define

$$(\{\tilde{\sigma}_i\}_{i\in[n]}, \{R_i\}_{i\in[n]}, \mathsf{com}_k) \leftarrow \mathsf{FSE.PSign}(\mathsf{sk}, \{m_i\}_{i\in[n]}; k)$$

Where $R_i = g^{r_i}$ for random values r_i, $\tilde{\sigma}_i = \frac{k + r_i + \mathsf{H}(R_i, m_i)\cdot \mathsf{sk}}{2}$, and $\mathsf{com}_k = g^k$. First, we assert that

$$\mathsf{FSE.PVerify}(\mathsf{pk}, \{m_i\}_{i\in[n]}, \{\tilde{\sigma}_i\}_{i\in[n]}, \{R_i\}_{i\in[n]}, \mathsf{com}_k) = 1$$

Which holds if and only if $g^{2\tilde{\sigma}_i} \overset{?}{=} \mathsf{com}_k \cdot R_i \cdot \mathsf{pk}^{c_i}$. Expanding this:

$$g^{2\tilde{\sigma}_i} = g^{k + r_i + \mathsf{H}(R_i, m_i)\cdot \mathsf{sk}} = g^k \cdot g^{r_i} \cdot (g^{\mathsf{sk}})^{\mathsf{H}(R_i, m_i)} = \mathsf{com}_k \cdot R_i \cdot \mathsf{pk}^{c_i}$$

Secondly, let $\{\sigma_i\}_{i\in[n]} := \mathsf{FSE.Recover}(\{\tilde{\sigma}_i\}_{i\in[n]}, \{R_i\}_{i\in[n]}, k)$, which implies that $\sigma_i = (2 \cdot \tilde{\sigma}_i - k, R_i)$. Therefore, the following holds:

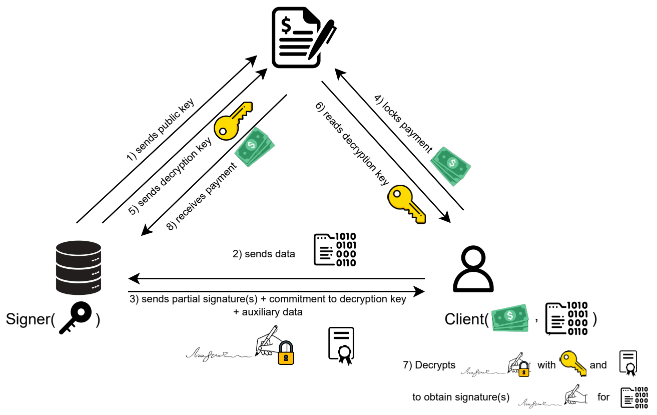


Fig. 6: Overview of the interaction between signer, client and the environment E, in a Fair signature exchange (FSE) protocol.

FSE.signer(k)		E	FSE.client(com$_k$, tok)
		$\xleftarrow[\text{for opening of com}_k]{\text{transaction with tokens tok}}$	
if $com_k \neq g^k$ abort	$\xrightarrow{k}$		
		if $g^k = \mathsf{com}_k$ then $\mathsf{tok}_s \leftarrow \mathsf{tok}$ else $\mathsf{tok}_s \leftarrow \perp$	
	$\xleftarrow{\mathsf{tok}_s}$		
retrieve tok$_s$			

Fig. 7: Interaction of signer and client with the transparent environment E. This can be implemented on Ethereum using a smart contract, and on Bitcoin using adaptor signatures, similar to [25].

$$\Pi.\mathsf{Verify}(\mathsf{pk}, m_i, \sigma_i) = 1 \iff g^{2\cdot\tilde{\sigma}_i - k} \overset{?}{=} R_i \cdot \mathsf{pk}^{c_i}$$

Furthermore, we can verify:

$$g^{2\cdot\tilde{\sigma}_i - k} = g^{r_i + \mathsf{H}(R_i, m_i)\cdot\mathsf{sk}} = g^{r_i}\cdot(g^{\mathsf{sk}})^{\mathsf{H}(R_i, m_i)} = R_i \cdot \mathsf{pk}^{c_i}$$

Finally since $\mathsf{com}_k = g^k$, according to the definition of transparent environment E, client having tokens tok, receives k and $\mathsf{tok}_s \neq \perp$. Thus, all three conditions hold with a probability of 1. □

Theorem 2. *The fair signature exchange scheme* FSE *described in Figure 4 satisfies signer fairness. In fact, for every* PPT *adversary $\mathcal{A}$, there exist* PPT *adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that,*

$$\mathsf{Adv}^{\mathrm{sf}}_{\mathcal{A}}(\lambda) \leq q_s \cdot \left(\mathsf{Adv}^{\mathrm{dl}}_{\mathcal{B}_1}(\lambda) + \frac{n\cdot q_H}{p}\right) + \mathsf{Adv}^{\mathrm{suf}}_{\mathcal{B}_2, Sch}(\lambda)$$

where $n = n(\lambda)$, $q_s = q_s(\lambda)$ and $q_H = q_H(\lambda)$ are upper bounds on the batch size, number of signing oracle queries, and number of random oracle queries made by $\mathcal{A}$ respectively.

Proof. We prove the security signer fairness game based on the *strong unforgeability* of Schnorr signatures and the DL assumption, as defined in Definitions 2 and 6, respectively. To provide intuition, assume that the adversary $\mathcal{A}$ outputs n_1 valid message/signature pairs while recovering only n_2 messages through $\mathsf{FSE.Recover}$, where $n_2 < n_1$. The adversary must have obtained a signature on a message m^* in one of the following ways:

– If $\mathcal{A}$ has not called $\mathsf{FSE.PSign}$ on m^*, we formally construct another adversary capable of breaking the unforgeability of the Schnorr signature scheme.
– If $\mathcal{A}$ has called $\mathsf{FSE.PSign}$ on m^* but has not called $\mathsf{FSE.Recover}$ on m^*, this implies that for a random g^k, computing the signature on m^* enables $\mathcal{A}$ to compute k. In this case, we formally construct an adversary capable of breaking the DL assumption.

Given the security of the Schnorr signature scheme (strong unforgeability) and the DL assumption, We conclude that, except with negligible probability, $\mathcal{A}$ cannot produce a signature for a message that has not been queried either to the signing oracle or to $\mathsf{FSE.Recover}$. This proves the signer fairness game. The formal proof can be found in Appendix A.1. $\square$

Theorem 3. *The fair signature exchange scheme* FSE *described in Figure 4 satisfies client fairness. In fact, every* PPT *adversary* $\mathcal{A}$ *wins* $\mathit{ClientFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda)$ *for scheme Figure 4 with an advantage of* 0.

Proof. Intuitively, given $\mathsf{com}_k = g^k$, client can verify the correctness of partial signatures by checking:

$$\mathsf{FSE.PVerify}(\{m_i\}_{i\in[n]}, \{\tilde{\sigma}_i\}_{i\in[n]}, \mathsf{aux}, \mathsf{com}_k) = 1$$
$$\iff g^{2\tilde{\sigma}_i} = \mathsf{com}_k \cdot R_i \cdot \mathsf{pk}^{c_i}.$$

By trusting E, client pays signer if and only if it receives the key k such that $\mathsf{com}_k = g^k$. This ensures that client makes the payment if and only if it can compute the correct signatures. The formal proof can be found in Appendix A.2. $\square$

5.2 Constructing Batch Adaptor Signatures from FSE

We define the notation of *batch adaptor signature scheme* $\mathsf{BAS}_{\Pi,\mathsf{Rel}}$ as an extension of adaptor signature scheme to sign multiple messages simultaneously, with respect to a signature scheme Π and a hard relation Rel is defined as a tuple of four PPT algorithms:

– $\{\tilde{\sigma}_i\} \leftarrow \mathsf{pBSign}(\mathsf{sk}, \{m_i\}, Y)$: The pre-signing algorithm takes as input the secret key for Π, a batch of messages, and a statement $Y \in L_{\mathsf{Rel}}$, and outputs partial signatures on all the messages.
– $b \leftarrow \mathsf{pBVrfy}(\mathsf{vk}, \{m_i\}, Y, \{\tilde{\sigma}_i\})$: The pre-verification algorithm is a deterministic algorithm which takes as input the public key for Π, a batch of messages, a statement Y and corresponding partial signatures $\{\tilde{\sigma}_i\}$, outputs a bit denoting whether or not all the partial signatures are valid.
– $\{\sigma_i\} \leftarrow \mathsf{Adapt}(\mathsf{vk}, \{\tilde{\sigma}_i\}, y)$: The adapting algorithm takes as input a batch of partial signatures, a witness y for the statement Y, and outputs full signatures.
– $y \leftarrow \mathsf{Extract}(\mathsf{vk}, \tilde{\sigma}_i, \sigma_i, Y)$: The extracting algorithm takes as input a partial signature, a corresponding full signature, a statement Y, then it either outputs a witness y such that $(Y, y) \in \mathsf{Rel}$ or $\perp$.

Similar to adaptor signatures [15], a batch adaptor signature scheme must satisfy the following properties:

– *Partial signature correctness.* This means that if the client and the server are both honest, then the client successfully gets valid signatures on all messages queried and the server learns the witnesses of all the corresponding instances.
– *Partial signature adaptability.* Informally, this means that if the partial signatures satisfy verification, i.e. pBVrfy outputs one, then the client gets valid signatures after running Adapt.
– *Unforgeability.* A malicious client should not be able to forge a signature.

– *Witness extractability.* This guarantees that a malicious client cannot use a partial signature for a message m, with respect to a statement Y to produce a valid signature σ without revealing a witness for Y.

Due to space restrictions, we do not formally define the notions listed above, but we will include them in the full paper. We now discuss how we can construct a Schnorr-signature-based batch adaptor scheme for the discrete log relation from our FSE construction. The core idea is to think of the encryption key k as the client's discrete log witness, and the same witness k is used to compute partial signatures for the whole batch of messages. More formally, when given the discrete log instance Y as input, the signer samples $\{r_i\}$ values as in standard Schnorr signing, then sets $\hat{R}_i$ to be g^{r_i}/Y for all i. The challenge values c_i's are computed using $\hat{R}_i$ instead of R_i, and then the signer simply returns $\frac{r_i+c_i\cdot\mathsf{sk}}{2}$. It is easy to see that the $\mathsf{FSE.PVerify}$ algorithm can directly be used to verify the partial signatures. The $\mathsf{FSE.Recover}$ algorithm can be used to adapt the partial signatures to get full signatures. Lastly, given any full signature $\sigma_i = (\hat{R}_i, s_i)$ and the corresponding partial signature $\hat{\sigma}_i$, the signer can easily compute the discrete log as $2\cdot\hat{\sigma}_i - s_i$. Figure 8 formally presents the batch adaptor construction.

```
BAS.pBSign($\mathsf{sk}, \{m_i\}, Y$)
1 :  $\forall i \in [n]$ :
2 :      $r_i \leftarrow\$ \mathbb{Z}_p$; $R'_i := g^{r_i}/Y$
3 :      $c_i := \mathsf{H}(R'_i, m_i)$
4 :      $s_i := \frac{r_i + c_i \cdot \mathsf{sk}}{2}$
5 :  return $(\{(s_i, R'_i)\}_{i\in[n]})$

BAS.Extract($\mathsf{vk}, \tilde{\sigma}_i, \sigma_i, Y$)
1 :  return $2 \cdot \tilde{\sigma}_i - s'_i$

BAS.pBVrfy($\mathsf{vk}, \{m_i\}_{i\in[n]}, Y, \{\tilde{\sigma}_i\}_{i\in[n]}$)
1 :  Parse $(s_i, R'_i) \leftarrow \tilde{\sigma}_i \forall i \in [n]$
2 :  $\forall\ i \in [n] : c_i := \mathsf{H}(R'_i, m_i)$
3 :  return true if $\forall\ i \in [n]$ :
4 :      $\mathsf{com}_i := R'_i \cdot \mathsf{pk}^{c_i}$
5 :      $g^{2s_i} \overset{?}{=} Y \cdot \mathsf{com}_i$

BAS.Adapt($\mathsf{vk}, \{\tilde{\sigma}_i\}_{i\in[n]}, y$)
1 :  $\forall i \in [n] : s'_i := 2 \cdot \tilde{\sigma}_i - y$
```

Fig. 8: The batch adaptor signature scheme BAS based on Schnorr signatures and the discrete log relation, constructed from our FSE scheme.

Analysis. Partial signature correctness for our construction is straightforward. Adaptability of the scheme directly follows from client fairness of the FSE scheme. Unforgeability can be proven by relying on signer fairness of FSE – the client cannot forge more signatures. Lastly, witness extractability holds based on the unforgeability of the underlying Schnorr signature scheme, similar to the proof in [9]. We will include formal proofs in the full version of the paper.

6 Implementation and Evaluation

To assess the performance of our protocol, we implemented it in Rust using the Arkworks framework [1]. Our source code is publicly available.* All experiments were conducted on a consumer-grade PC equipped with an Intel i5-7200U CPU (2 cores) and 8 GB of RAM. Our simple baseline for measurement is the original Schnorr protocol. To measure the computation of the signer, we consider the running time of FSE.PSign algorithm in Figure 4, and for the client, we measure the running time of the algorithms FSE.PVerify and FSE.Recover. We find that, compared to the original Schnorr protocol, our protocol has only 25% overhead for the signer for $n > 8$ signatures (and upto $0.9x$ for smaller values of n); and under 2x overhead for the verifier (Figure 10). We evaluate our scheme on two curves: BLS12-381 (pairing-friendly) and Vesta (non-pairing-friendly). Notably, initiating our scheme on Vesta yields a 2-3x performance improvement compared to BLS12-381, which is advantageous as we do not require pairing.

Computation comparison to general-purpose FDE [25]. We benchmark our protocol against the state-of-the-art FDE schemes presented in [25], which we refer to as KZG-Paillier and KZG-Elgamal (Figure 9). For this comparison, we assume that the transferred data consists of a vector of field elements. A single Schnorr signature comprises a randomly chosen group element R and a secret field element s. We assume s is transmitted through FDE, allowing us to equate a signature with a field element and assume R is transmitted in plaintext. Similar to the original experiment in [25], we measure the time required to generate proofs by the prover and verify proofs by the verifier. The FDE scheme employs KZG polynomial commitment on the pairing-friendly BLS12-381 curve, introducing additional overhead. It is important to note that the FDE scheme includes additional costly operations, such as data encryption and KZG commitment generation, which are not accounted for in their performance evaluation but are irrelevant to our protocol and, therefore, not included in our comparison. For the sake of comparison, we use BLS12-381 to initialize our scheme too. As seen in Figure 9, the signer in our FSE protocol is almost 10x more efficient than the KZG-Paillier prover; relative to the KZG-Elgamal prover, our signer almost has the same time for $n = 2^{10}$ but much faster some smaller n. The verifier in our FSE scheme is about 10x more efficient than the KZG-Paillier verifier and 100x more efficient than the verifier in KZG-Elgamal. Hence, our FSE scheme in general is orders of magnitude more efficient than the general purpose FDE constructions in [25].

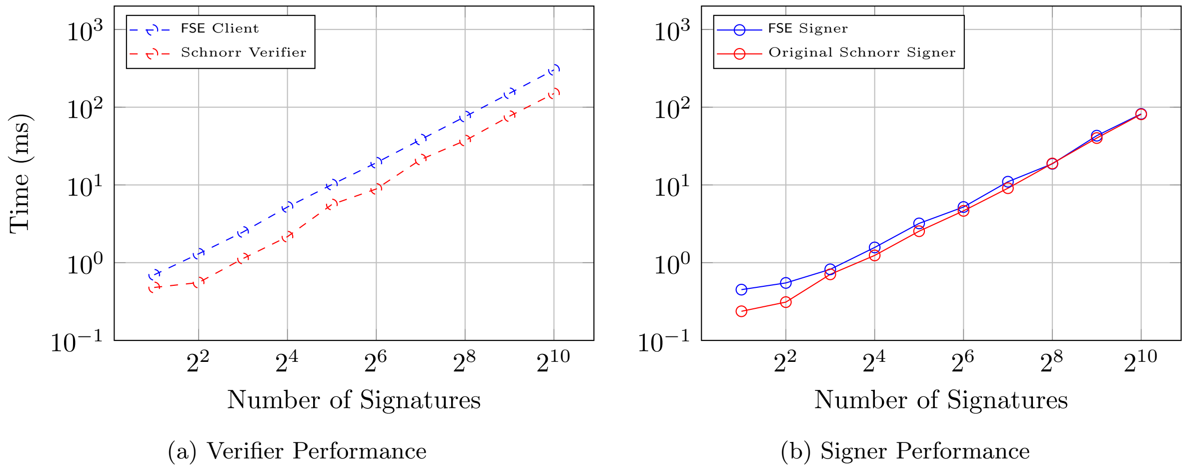
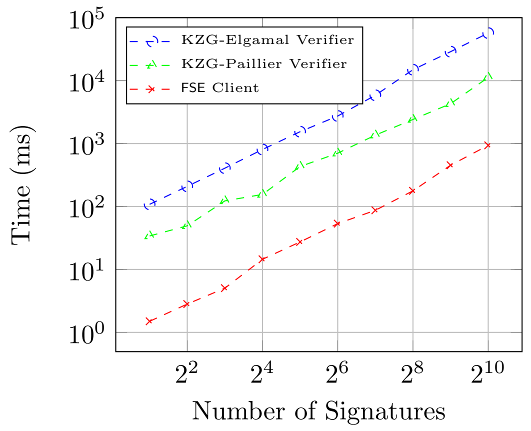
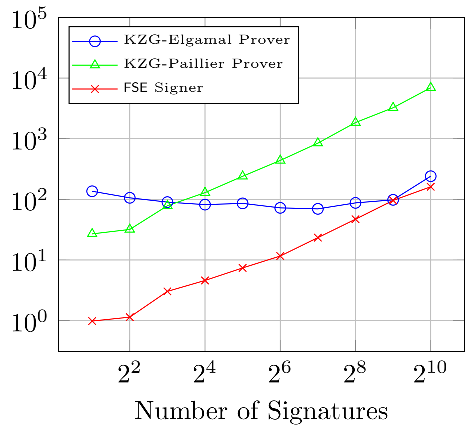


Fig. 10: Comparison of Verifier and Signer performance between FSE scheme and Original Schnorr, both instantiated with Vesta curve. FSE has almost no overhead on the signer but has a 2x overhead on the verifier (client) compared to the original Schnorr scheme.

*https://github.com/h-hafezi/fair_schnorr_signature_exchange



(a) Verifier Performance



(b) Signer Performance

Fig. 9: Comparison between our FSE scheme and two schemes in [25], all instantiated with BLS12-381. (a) Verifier Comparison, (b) Prover Comparison. The verifier in our FSE scheme is about 10x more efficient than KZG-Paillier verifier and 100x more efficient than KZG-Elgamal. Our prover is about 10x more efficient than the KZG-Paillier prover, while compared to the KZG-Elgamal prover, it is faster for small n and almost has the same running time for $n = 2^{10}$.

Communication comparison to [25]. Communication using blockchains is extremely expensive. At the time of writing, publishing 1MB of data on the Ethereum blockchain costs approximately \$17,000 USD. Therefore, minimizing on-chain communication is crucial. Our protocol, like [25], operates FSE for n signatures with constant on-chain communication: only a single group element (com_k) and one scalar field element (k) are published on E, totalling just 136 bytes with BLS12-381. Unlike [25], our protocol also minimizes off-chain communication. To exchange n signatures (beyond the messages to be signed), the signer and client only exchange n field elements $\{\tilde{\sigma}_i\}_{i\in[n]}$ and $n+1$ group elements: $\{R_i\}_{i\in[n]}$ along with com_k. This contrasts sharply with [25], which incurs significant bandwidth overheads, with constants 10x and 50x larger for KZG-Elgamal and KZG-Paillier, respectively, relative to the data size.

7 Fair Signature Exchange for Hidden Messages

We extend our protocol from the last section to allow exchanging blind signatures, i.e., signatures on hidden messages. We start with formally defining blind signatures. In addition, we propose a construction based on blind Schnorr signatures, but we do not provide a formal proof.

7.1 Blind signatures

A blind signature scheme allows a user to obtain a signature from a signer on a message m in such a way that (i) the signer is unable to recognize the signature later (*blindness*, which in particular implies that m remains hidden from the signer) and (ii) the user cannot compute more signatures than issued by the signer (*unforgeability*). Blind signatures have a wide range of applications e.g. e-cash [6, 7, 20, 5, 13], e-voting [14, 17] and anonymous credentials [4, 2]. We use the formal definition and construction of blind signatures from [12], which can also be found in Appendix B. Figure 11 presents the blind signing protocol. The other algorithms, $\mathsf{BS.Setup}$, $\mathsf{BS.Gen}$, and $\mathsf{BS.Verify}$, are defined similarly to the standard Schnorr signature scheme. This scheme is secure with arbitrary polynomially many concurrent sessions assuming the hardness of the one-more discrete log assumption (OMDL) and the modified ROS assumption (MROS) in the algebraic group model (AGM) [11].

$\underline{\mathsf{BS.Sign}(\mathsf{sk})}$		$\underline{\mathsf{BS.user}(\mathsf{pk}, m)}$
$r_0,\, r_1 \leftarrow\!\$\; \mathbb{Z}_p$		
$R_0 := g^{r_0},\, R_1 := g^{r_1}$	$\xrightarrow{\quad R_0,\, R_1 \quad}$	
		$\alpha_0,\, \alpha_1,\, \beta_0,\, \beta_1 \leftarrow\!\$\; \mathbb{Z}_p$
		$R'_0 := R_0 \cdot g^{\alpha_0} \cdot \mathsf{pk}^{\beta_0}$
		$R'_1 := R_1 \cdot g^{\alpha_1} \cdot \mathsf{pk}^{\beta_1}$
		$c'_0 := \mathsf{H}(R'_0, m),\, c'_1 := \mathsf{H}(R'_1, m)$
	$\xleftarrow{\quad c_0,\, c_1 \quad}$	$c_0 := c'_0 + \beta_0;\, c_1 := c'_1 + \beta_1$
$b \leftarrow\!\$\; \{0,1\}$		
$s \leftarrow r_b + c_b \cdot \mathsf{sk}$	$\xrightarrow{\quad s,\, b \quad}$	**if** $g^s \neq R_b \cdot \mathsf{pk}^{c_b}$ **then return** $\perp$
		return $\sigma \leftarrow (R'_b,\, s' := s + \alpha_b)$

Fig. 11: Clause blind Schnorr signature signing protocol

7.2 Fair blind signature exchange (FBSE)

We define FBSE similar to FSE. The two main differences are, (1) the signing protocol is now interactive and (2) the protocol is defined on top of a blind signature scheme BS, instead of a non-blind signature scheme Π. Formally, a *Blind Fair Signature Exchange* (FBSE) signature protocol is a protocol between client and signer involving a transparent payment environment E. It consists of PPT algorithms (Setup, KeyGen, $\mathsf{BlindPSign}$, Verify, $\mathsf{Execute}$, $\mathsf{Recover}$), the protocol is initialized with a blind signature scheme BS, where client has a public key pk and signer has the corresponding secret key sk. Both these parties are assumed to have access to unbiased random coins and the functionalities of BS.

– $\mathsf{FBSE.Setup}(1^\lambda) \rightarrow \mathsf{pp}$. Probabilistic polynomial-time algorithm that outputs the public parameters for the system (e.g., the description of appropriate spaces). All the following algorithms and protocols implicitly take the public parameters; we omit them for brevity.

- $\mathsf{FBSE.KeyGen}() \to (\mathsf{pk}, \mathsf{sk})$ is a randomized algorithm that samples a secret signing key sk and computes the corresponding public key pk.
- $\mathsf{FBSE.BlindPSign}\big(\langle \mathsf{client}(\mathsf{pk}, \{m_i\}_{i\in[n]}) \leftrightarrow \mathsf{signer}(\mathsf{sk})\rangle\big)$. Parties client and signer engage in an interactive protocol, where at the end client receives partial signatures $\{\tilde{\sigma}_i\}_{i\in[n]}$ on the messages along with auxiliary data aux and commitment com_k. For a 3-round protocol, the interaction can be realized by the following algorithms:
$$\begin{aligned}
(\mathsf{msg}_{\mathsf{client},0},\ \mathsf{st}_{\mathsf{client},0}) &\leftarrow \mathsf{FBSE.client}_0(\mathsf{pk},\ \{m_i\}_{i\in[n]})\\
(\mathsf{msg}_{\mathsf{signer},1},\ \mathsf{st}_{\mathsf{signer}}) &\leftarrow \mathsf{FBSE.signer}_1(\mathsf{sk},\ \mathsf{msg}_{\mathsf{client},0})\\
(\mathsf{msg}_{\mathsf{client},1},\ \mathsf{st}_{\mathsf{client},1}) &\leftarrow \mathsf{FBSE.client}_1(\mathsf{st}_{\mathsf{client},0},\ \mathsf{msg}_{\mathsf{signer},1})\\
(\mathsf{msg}_{\mathsf{signer},2},\ k) &\leftarrow \mathsf{FBSE.signer}_2(\mathsf{st}_{\mathsf{signer},1},\ \mathsf{msg}_{\mathsf{client},1})\\
(\{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ \mathsf{com}_k) &\leftarrow \mathsf{FBSE.client}_2(\mathsf{st}_{\mathsf{client},1},\ \mathsf{msg}_{\mathsf{signer},2})
\end{aligned}$$
Typically, $\mathsf{FBSE.client}_0$ just initiates the session, and thus $\mathsf{msg}_{\mathsf{client},0} = ()$ and $\mathsf{st}_{\mathsf{client},0} = (\mathsf{pk},\ \{m_i\}_{i\in[n]},\ n)$.
- $\mathsf{FBSE.PVerify}(\mathsf{pk},\ \{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ \mathsf{com}_k) \to \{0,1\}$ is a deterministic algorithm that on input public key pk, partial signatures $\{\tilde{\sigma}_i\}_{i\in[n]}$, auxiliary data aux and commitment com_k, checks whether the partial signatures are formatted correctly.
- $\mathsf{FBSE.Execute}(\langle \mathsf{client}(\mathsf{com}_k,\ \mathsf{tok}) \leftrightarrow \mathsf{E} \leftrightarrow \mathsf{signer}(k)\rangle)$. client and signer communicate independently with a transparent third-party environment E, in which at the end, client outputs k which is opening to com_k and signer outputs token tok. This two-round interaction can be realized with the following algorithms, $\mathsf{FBSE}^{(E)}$ indicates the algorithm having access to the transparent environment E:
$$\begin{aligned}
\mathsf{st}_{\mathsf{E},0} &\leftarrow \mathsf{FBSE}^{(\mathsf{E})}.\mathsf{Init}()\\
\mathsf{st}_{\mathsf{E},1} &\leftarrow \mathsf{FBSE}^{(\mathsf{E})}.\mathsf{ExeClient}_0(\mathsf{com}_k)\\
(\mathsf{tok}, \mathsf{st}_{\mathsf{E},2}) &\leftarrow \mathsf{FBSE}^{(\mathsf{E})}.\mathsf{ExeSigner}_0(k)
\end{aligned}$$
If the interaction is successful then $\mathsf{tok} \neq \perp$ otherwise $\mathsf{tok} = \perp$.
- $\mathsf{FBSE.Recover}(\{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ k)$. On input partial signatures $\{\tilde{\sigma}_i\}_{i\in[n]}$, decryption key k and auxiliary data aux, outputs signatures $\{\sigma_i\}_{i\in[n]}$.

Since the definition has been revised, the security definitions must be updated, particularly in light of the new constraints, e.g. allowing interaction in the signing protocol. We introduce a new definition for client privacy, where the goal is to ensure that the signer learns nothing about the client's messages being signed. The updated security definitions are provided in Appendix B.1.

Our FBSE construction. We observe that our original FSE in Figure 4, can be modified to also work for exchanging blind signatures. Apart from the signing algorithm, the Setup, KeyGen, Verify and Recover algorithms remain the same as in Figure 4. For signing, we tweak the interactive protocol from Figure 11 in the same way as the non-blind Schnorr signature, for example, client first receives the partial signatures $\{\tilde{\sigma}_i\}$ instead of sending $\{s_i\}$ directly, and later client can compute values $\{s_i\}$ by obtaining the key k. Figure 12 formally describes the signing algorithm for our blind FSE scheme FBSE, in which at the end, client receives the partial signatures. Appendix B.4 formally analyses the client privacy and client fairness for our FBSE scheme. We remark that we do not have a formal proof for signer fairness, but we conjecture that it holds based on the one-more discrete log assumption.

7.3 Application of FBSE

Here we point out two interesting applications of FBSE:

Decentralized privacy pass. Privacy pass [8] is a cryptographic protocol that enables users to bypass CAPTCHAs while preserving privacy using anonymous, one-time-use tokens. Each token is blindly signed to keep serial numbers hidden during issuance, ensuring privacy when the token is redeemed. While centralized entities like Cloudflare currently issue such tokens, this approach risks censorship and single-point-of-failure.

```
signer((p, G, g, H), sk)                                              client((p, G, g, H), pk, {m_i}_{i∈[n]})
$\forall\, i \in [n] \;:\; r_{i,0},\, r_{i,1} \leftarrow\!\$\, \mathbb{Z}_p$
    $R_{i,0} \leftarrow g^{r_{i,0}}$
    $R_{i,1} \leftarrow g^{r_{i,1}}$
                          $\xrightarrow{\mathsf{aux}_0 \,:=\, \{R_{i,0},\ R_{i,1}\}_{i\in[n]}}$
                                                                      $\forall i \in [n] \;:$
                                                                          $\alpha_{i,0},\, \beta_{i,0},\, \alpha_{i,1},\, \beta_{i,1} \leftarrow\!\$\, \mathbb{Z}_p$
                                                                          $R'_{i,0} := R_{i,0} \cdot g^{\alpha_{i,0}} \cdot \mathsf{pk}^{\beta_{i,0}}$
                                                                          $R'_{i,1} = R_{i,1} \cdot g^{\alpha_{i,1}} \cdot \mathsf{pk}^{\beta_{i,1}}$
                                                                          $c'_{i,0} = \mathsf{H}(R'_{i,0},\, m)$
                                                                          $c'_{i,1} = \mathsf{H}(R'_{i,1},\, m)$
                                                                          $c_{i,0} = c'_{i,0} + \beta_{i,0} \mod p$
                                                                          $c_{i,1} = c'_{i,1} + \beta_{i,1} \mod p$
                          $\xleftarrow{\{c_{i,0},\, c_{i,1}\}_{i\in[n]}}$
$\forall\, i \in [n] : b_i \leftarrow\!\$\, \{0,\, 1\}$
$k \leftarrow\!\$\, \mathbb{Z}_p \;;\; \mathsf{com}_k \leftarrow g^k$
$s_i := r_{i,b_i} + c_{i,b_i} \cdot \mathsf{sk} \mod p$
$\tilde{\sigma}_i := \dfrac{k + s_i}{2}$
                          $\xrightarrow{\{\tilde{\sigma}_i\}_{i\in[n]},\, \{b_i\}_{i\in[n]},\, \mathsf{com}_k}$
                                                                      $\mathsf{aux} := \{R'_{i,b_i}\}_{i\in[n]}$
                                                                      return $(\{\tilde{\sigma}_i + \alpha_{i,b_i}/2\}_{i\in[n]},\, \mathsf{aux},\, \mathsf{com}_k)$
```

Fig. 12: The interactive signing protocol for our fair blind signature exchange scheme FBSE

A decentralized marketplace for anonymous tokens could address these concerns, allowing diverse entities to issue tokens anonymously. To enable this, blind fair signature exchange protocol incentivizes providers by allowing clients to obtain tokens in exchange for payment. This interactive system ensures privacy, fairness, and a more robust decentralized credential ecosystem.

Automated coin shuffling. Consider a smart contract (sc) that, upon receiving a serial number and a valid signature on it, verifies the serial number's freshness and transfers \$1 to a designated account. However, the operator of sc cannot trust users to provide valid signatures in advance, nor can users deposit funds beforehand. Additionally, the operator must remain unaware of the specific serial numbers it signs. By leveraging FBSE, mutual trust between the operator and the client is unnecessary, the blindness property ensures that sc does not learn the signed serial numbers, and finally the underlying batching property ensures minimized communication on the blockchain. This mechanism provides privacy within the pool, proportional to its size.

8 Conclusion

Fair Signature Exchange (FSE) is a novel cryptographic primitive inspired by Fair Data Exchange (FDE). It generalizes the concept of adaptor signatures by extending their functionality to support batching, enabling conditional and atomic exchange of multiple signatures in a single operation. FSE has significant applications in decentralized finance (DeFi), including batch atomic swaps and bridges. FSE extends the definition of adaptor signatures to consider the blockchain in the security definitions as well as batching. In this work, we formally define FSE and provide comprehensive security definitions as well as a construction based on Schnorr signatures. Our work includes rigorous proofs of security, showing that FSE achieves its goals while minimizing communication with the trusted third party (e.g., blockchain) and does not rely on heavy cryptographic primitives like SNARKs. This makes FSE a lightweight and efficient solution for fair signature exchanges in decentralized ecosystems.

8.1 Future work

Fair blind signature exchange. We proposed a definition for fair blind signature exchange with practical motivations such as coin shuffling and fair exchange of privacy passes. Finally, we proposed a construction based on the Schnorr blind signature, deferring a complete security proof to future work.

Fair proof exchange. Proof systems are useful tools that help outsource computation trustlessly with tremendous applications such as roll-ups. One interesting question is how to construct efficient *fair proof exchange* which can be the building block for a *proof marketplace.* Distributed proof systems [23, 18, 27] involve a coordinator that partitions a computation into smaller chunks. These chunks are then assigned to worker nodes which compute proofs for their respective chunks. Subsequently, the coordinator aggregates the individual chunk proofs to produce a proof for the entire computation. Fair proof exchange mechanisms can incentivize worker nodes for their contributions, while ensuring they are paid if they provide the agreed-upon proofs.

Acknowledgments. We thank Ali Yazdizadeh for his assistance with a technical issue. The initial stages of this project were conducted while Sourav Das and Aditi Partap were at a16z Crypto Research. This work was supported by a16z Crypto. Joseph Bonneau was also supported by DARPA Agreement HR00112020022 and NSF Grant CNS-2239975. The views and conclusions contained in this material are those of the authors and do not necessarily reflect the official policies or endorsements of the United States Government, DARPA, a16z Crypto, or any other supporting organization.

References

1. arkworks contributors. `arkworks` zksnark ecosystem, 2022.
2. Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. Cryptology ePrint Archive, Paper 2012/298, 2012.
3. Dan Boneh and Victor Shoup. A graduate course in applied cryptography, 2023. Cryptobook.
4. Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, page 302–318, Berlin, Heidelberg, 1993. Springer-Verlag.
5. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. Cryptology ePrint Archive, Paper 2005/060, 2005.
6. David Chaum. Blind signatures for untraceable payments. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology Proceedings of Crypto 82*, pages 199–203, 1983.
7. David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Annual International Cryptology Conference*, 1990.
8. Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PETS*, 2018.
9. Andreas Erwig, Sebastian Faust, Kristina Hostáková, Monosij Maitra, and Siavash Riahi. Two-party adaptor signatures from identification schemes. Cryptology ePrint Archive, Paper 2021/150, 2021.
10. Muhammed F. Esgin, Oguzhan Ersoy, and Zekeriya Erkin. Post-quantum adaptor signatures and payment channel networks. Cryptology ePrint Archive, Paper 2020/845, 2020.
11. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. Cryptology ePrint Archive, Paper 2017/620, 2017.
12. Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed elgamal encryption in the algebraic group model. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pages 63–95. Springer, 2020.
13. Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Transferable constant-size fair e-cash. Cryptology ePrint Archive, Paper 2009/146, 2009.
14. Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, ASIACRYPT '92, page 244–251, Berlin, Heidelberg, 1992. Springer-Verlag.
15. Paul Gerhart, Dominique Schröder, Pratik Soni, and Sri AravindaKrishnan Thyagarajan. Foundations of adaptor signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 161–189. Springer, 2024.
16. Philipp Hoenisch, Subhra Mazumdar, Pedro Moreno-Sanchez, and Sushmita Ruj. LightSwap: An atomic swap does not require timeouts at both blockchains. Cryptology ePrint Archive, Paper 2022/1650, 2022.
17. Marcin Kucharczyk. Blind signatures in electronic voting systems. In *International Conference on Computer Networks*, 2010.
18. Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkRollups via fully distributed zero-knowledge proofs. Cryptology ePrint Archive, Paper 2023/1271, 2023.
19. Arash Mirzaei, Amin Sakzad, Jiangshan Yu, and Ron Steinfeld. Daric: A storage efficient payment channel with penalization mechanism. Cryptology ePrint Archive, Paper 2022/1295, 2022.
20. Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, page 324–337, Berlin, Heidelberg, 1991. Springer-Verlag.
21. Henning Pagnia and Felix C. Gartner Darmstadt. On the impossibility of fair exchange without a trusted third party. 1999.
22. Siavash Riahi and Orfeas Stefanos Thyfronitis Litos. Bitcoin clique: Channel-free off-chain payments using two-shot adaptor signatures. Cryptology ePrint Archive, Paper 2024/025, 2024.
23. Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. Cryptology ePrint Archive, Paper 2024/1208, 2024.
24. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, New York, USA.

25. Ertem Nusret Tas, István András Seres, Yinuo Zhang, Márk Melczer, Mahimna Kelkar, Joseph Bonneau, and Valeria Nikolaenko. Atomic and fair data exchange via blockchain. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, 2024.
26. Nikhil Vanjani, Pratik Soni, and Sri AravindaKrishnan Thyagarajan. Functional adaptor signatures: Beyond all-or-nothing blockchain-based payments. Cryptology ePrint Archive, Paper 2024/1523, 2024.
27. Wenhao Wang, Fangyan Shi, Dani Vilardell, and Fan Zhang. Cirrus: Performant and accountable distributed SNARK. Cryptology ePrint Archive, Paper 2024/1873, 2024.

A Deferred Proofs

We now prove that our proposed scheme in Figure 4 satisfies signer-fairness and client-fairness.

A.1 Server fairness

Definition 6 (Discrete Logarithm Assumption). *We say the discrete logarithm assumption (DL) holds with respect to group generator* $\mathsf{GGen}(1^\lambda)$ *if for any PPT adversary* $\mathcal{A}$*, given* $(\mathbb{G}, p, g) \leftarrow \mathsf{GGen}(1^\lambda)$ *and* $x \leftarrow\!\!\$\ \mathbb{Z}_p$*,* $\mathcal{A}(\mathbb{G}, p, g, g^x)$ *outputs* x *with negligible probability.*

Proof of Theorem 2. Let $\mathcal{L} = \{m_i^*, \sigma_i^*\}_{i\in[t']}$ denote the list of signatures returned by $\mathcal{A}$ in the signer fairness game. Let $\sigma_i^* = (R_i^*, s_i^*)$ be the signatures returned by $\mathcal{A}$, for all $i \in [t']$. Let M denote the list of messages for which $\mathcal{A}$ queried the SIGN oracle. Let $\mathsf{R} : \mathcal{M} \to \mathbb{G}$ be a map storing the R_i values output by the challenger in response to a SIGN query by $\mathcal{A}$. Let $\mathsf{B} : \mathcal{M} \to \mathbb{N}$ be a map storing the batch number (or session id) corresponding to the messages for which the SIGN oracle was called. Lastly, let X be a list of session ids for which the $\mathsf{EXECUTE}$ is called, and it does not return $\perp$. Let E_1 be the event that there exists a message - signature pair $(m^*, \sigma^* = (R^*, s^*))$ such that (a) either $m^* \notin \mathsf{M}$, meaning that the adversary never called the SIGN oracle with m^* as one of the inputs, (b) or, $m^* \in \mathsf{M}$, $R^* \neq \mathsf{R}[m^*]$, and $\mathsf{B}[m^*]$ is not in X. Additionally, let E_2 denote the event that there exists a message - signature pair $(m^*, \sigma^* = (R^*, s^*))$ such that $m^* \in \mathsf{M}$, the corresponding nonce value R^* is equal to $\mathsf{R}(m^*)$, and, $\mathsf{B}[m^*]$ is not in X. We note that in the event $\neg\mathsf{E}_1 \wedge \neg\mathsf{E}_2$, the number t' of signatures output by $\mathcal{A}$ cannot be greater than t because for all (m, σ) pairs output by the adversary, both the SIGN and EXEC oracles were called and successfully run. Hence, the adversary can win the signer fairness game only if either E_1 or E_2 occurs. This implies that,

$$\begin{aligned}
&\Pr[\mathsf{SignerFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda) = 1] &(1)\\
&= \Pr[\mathsf{SignerFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda) \ \wedge\ \mathsf{E}_1] &(2)\\
&+ \Pr[\mathsf{SignerFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda) \ \wedge\ \mathsf{E}_2] &(3)
\end{aligned}$$

Lemmas 1 and 2 below prove the theorem.

Lemma 1. *There exists a* PPT *adversary* $\mathcal{B}_1$ *such that,*

$$\Pr[\mathsf{SignerFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda) = 1 \wedge \mathsf{E}_1] \leq \mathsf{Adv}^{\mathrm{suf}}_{\mathcal{B}_1, Sch}(\lambda)$$

Proof. We construct an adversary $\mathcal{B}_1$ for the strong unforgeability of Schnorr as follows. It receives $\mathsf{pp} \leftarrow (\mathbb{G}, g, p, \mathsf{H}), \mathsf{pk}$ from its challenger and simulates the $\mathsf{SignerFairness}$ game to $\mathcal{A}$. It first sends pp, pk to $\mathcal{A}$. It maintains maps M, R and B as described above – they are all initialized as $\perp$. It then responds to the oracle queries as follows:

- $\mathsf{SIGN}(\{m_i\}_{i\in[n]})$. As in the signer fairness game, $\mathcal{B}_1$ increments the counter c by one, and samples a random key $k \leftarrow\!\!\$\ \mathbb{Z}_p$. It then queries its unforgeability challenger for signatures on m_i for all $i \in [n]$. Let $\{(R_i, s_i)\}_{i\in[n]}$ be the challenger's response. It computes $\mathsf{com}_k \leftarrow g^k$, and $\tilde{\sigma}_i = (k + s_i)/2$ for all $i \in [n]$. It stores k and n in the maps $\mathcal{K}$ and $\mathcal{N}$ respectively. Additionally, it sets $\mathsf{M} \leftarrow \mathsf{M} \cup \{m_i\}_{i\in[n]}$, $\mathsf{R}[m_i] \leftarrow R_i$ and $\mathsf{B}[m_i] \leftarrow c$ for all $i \in [n]$. It then returns $\{\tilde{\sigma}_i\}_{i\in[n]}$, com_k and $\mathsf{aux} = \{R_i\}_{i\in[n]}$ to $\mathcal{A}$.
- $\mathsf{EXEC}(j, \mathsf{com}_k)$. $\mathcal{B}_1$ aborts if $j \notin \mathcal{K}$ or if $\mathsf{com}_k \neq com(\mathcal{K}[j])$, as in the signer fairness game. Otherwise, it increments t by n, adds j to X and outputs $\mathcal{K}[j]$.
- $\mathsf{H}(R, m)$. It queries its challenger for $\mathsf{H}(R, m)$ and forwards the response to $\mathcal{A}$.

Eventually, $\mathcal{A}$ outputs a list of message - signature pairs, $\mathcal{L} = \{m_i^*, \sigma_i^*\}_{i\in[t']}$. $\mathcal{B}_1$ finds a message-signature pair $(m^*, (R^*, s^*)) \in \mathcal{L}$ such that, either $m^* \notin \mathsf{M}$ or, $m^* \in \mathsf{M}$ but $R^* \neq \mathsf{R}[m^*]$ and $\mathsf{B}[m^*]$ is not in X. It aborts if it cannot find any such message. Otherwise, it returns $(m^*, (R^*, s^*))$ to its challenger. Now, we argue that if $\mathcal{A}$ wins the signer fairness game, and if E_1 occurs, then $\mathcal{B}_1$ also wins its unforgeability game.

This is because, conditioned on E_1, $\mathcal{B}_1$ will not abort because there exists a message in $\mathcal{L}$ that satisfies all the conditions listed above. Additionally, if $m^* \notin \mathsf{M}$, then, $\mathcal{B}_1$ never queried its challenger for a signature on m^*. Otherwise, if $m^* \in \mathsf{M}$ but $R^* \neq \mathsf{R}[m^*]$, then, the signature (R^*, s^*) on m^* output by $\mathcal{A}$ is different from the signature $(\mathsf{R}[m^*], \cdot)$ output by the unforgeability challenger. In both cases, $\mathcal{B}$ successfully outputs a valid forgery. This proves the lemma. □

Lemma 2. *There exists a* PPT *adversary* $\mathcal{B}_2$ *such that,*

$$\frac{1}{q_s} \cdot \Pr[\mathsf{SignerFairness}_{\mathcal{A}}^{\mathsf{FSE}[\Pi]}(1^\lambda) = 1 \wedge \mathsf{E}_2] - \frac{n \cdot q_H}{p} \leq \mathsf{Adv}_{\mathcal{B}_2}^{\mathrm{dl}}(\lambda)$$

where $q_s = q_s(\lambda)$, $n = n(\lambda)$ and $q_H = q_H(\lambda)$ are upper bounds on the number of SIGN *queries, the batch size, and the number of random oracle queries issued by $\mathcal{A}$ respectively.*

Proof. We construct a discrete log adversary $\mathcal{B}_2$ as follows. It gets as input $(\mathbb{G}, g, h, p)$, where p is the order of the group $\mathbb{G}$, and g is a generator. It plays the role of challenger to $\mathcal{A}$ in the SignerFairness game as follows. First, it samples $\mathsf{sk} \leftarrow\!\!\$\; \mathbb{Z}_p$ and sets $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$. It then sends $\mathsf{pp} = (\mathbb{G}, g, p), \mathsf{pk}$ to $\mathcal{A}$. To respond to the oracle queries, $\mathcal{B}_2$ maintains some metadata. Specifically, it stores a map $H : \mathbb{G} \times \mathcal{M} \to \mathbb{Z}_p$ to handle random oracle queries. It initializes another map $\mathsf{A} : \mathcal{M} \to \mathbb{Z}_p$ to store $\tilde{\sigma}$ values used for responding to SIGN queries. It also maintains M, R and B as described above. Additionally, let q_s be an upper bound on the number of signing oracle queries issued by $\mathcal{A}$. $\mathcal{B}_2$ samples $i^* \leftarrow\!\!\$\; [q_s]$, as its guess for the signing session containing a forgery message. We now describe how $\mathcal{B}_2$ responds to oracle queries:

- $\mathsf{H}(R, m)$. If $(R, m) \in H$, then simply output $H(R, m)$. Otherwise, it samples $c \leftarrow\!\!\$\; \mathbb{Z}_p$, sets $H(R, m) \leftarrow c$ and returns c to $\mathcal{A}$.
- $\mathsf{SIGN}(\{m_i\}_{i\in[n]})$. $\mathcal{B}$ increments c as in the game. It sets $\mathsf{M} \leftarrow \mathsf{M} \cup \{m_i\}_{i\in[n]}$ and $\mathsf{B}[m_i] \leftarrow c$ for all $i \in [n]$.
  - If $c \neq i^*$, $\mathcal{B}_2$ honestly runs the signature protocol. More formally, for each $i \in [n]$, it samples $r_i \leftarrow\!\!\$\; \mathbb{Z}_p$, computes $R_i \leftarrow g^{r_i}$. It queries $c_i \leftarrow \mathsf{H}(R_i, m_i)$ for all $i \in [n]$, and then computes $s_i \leftarrow r_i + c_i \cdot \mathsf{sk}$. It samples $k_c \leftarrow\!\!\$\; \mathbb{Z}_p$, computes $\mathsf{com}_k \leftarrow g^{k_c}$ and stores $\mathcal{K}[c] \leftarrow k_c$. It computes $\tilde{\sigma}_i \leftarrow (k_c + s_i)/2$ and $\mathsf{com}_i \leftarrow g^{s_i}$ for all $i \in [n]$. It then sets $\mathsf{R}[m_i] \leftarrow R_i$ for all $i \in [n]$, and sends $\{\tilde{\sigma}_i\}_{i\in[n]}$, $\mathsf{aux} = \{R_i\}_{i\in[n]}$ and com_k to $\mathcal{A}$.
  - If $c = i^*$, $\mathcal{B}$ sets $\mathsf{com}_k^* \leftarrow h$.
    - ∗ It samples $\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n \leftarrow\!\!\$\; \mathbb{Z}_p$.
    - ∗ It then computes $\mathsf{com}_i^* \leftarrow g^{2\tilde{\sigma}_i}/h$, and samples $c_i \leftarrow\!\!\$\; \mathbb{Z}_p$ for all $i \in [n]$.
    - ∗ It computes $R_i^* \leftarrow \mathsf{com}_i^*/(\mathsf{pk})^{c_i}$ for all $i \in [n]$.
    - ∗ If, for any i, $(R_i^*, m_i) \in H$, $\mathcal{B}_2$ aborts. Otherwise, it sets $H(R_i^*, m_i) = c_i$ for all $i \in [n]$.
    - ∗ It sets $\mathsf{A}[m_i] \leftarrow \tilde{\sigma}_i$ for all $i \in [n]$.
    - ∗ Lastly, it sends $\{\tilde{\sigma}_i\}_{i\in[n]}$, $\mathsf{aux} = \{R_i^*\}_{i\in[n]}$ and com_k^* to $\mathcal{A}$.
- $\mathsf{EXECUTE}(j, \mathsf{com}_k)$. As in the signer fairness game, if $j \notin \mathcal{K}$ or if $j \neq i^*$ and $\mathsf{com}_k \neq g^{\mathcal{K}[j]}$, or if $j = i^*$ and $\mathsf{com}_k \neq h$, then, $\mathcal{B}_2$ returns $\perp$. Otherwise, if $j = i^*$ and $\mathsf{com}_k = h$, $\mathcal{B}_2$ aborts. Otherwise, $\mathcal{B}_2$ adds j to X and simply returns $\mathcal{K}[j]$.

Eventually, $\mathcal{A}$ outputs a list of message - signature pairs, $\mathcal{L} = \{m_i^*, \sigma_i^*\}_{i\in[t']}$. $\mathcal{B}_2$ finds a message-signature pair $(m^*, (R^*, s^*)) \in \mathcal{L}$ such that, $m^* \in \mathsf{M}$, the corresponding nonce value R^* is equal to $\mathsf{R}(m^*)$, and, $\mathsf{B}[m^*]$ is not in X. It aborts if it cannot find any such message, or if $\mathsf{B}[m^*] \neq i^*$. Otherwise, let $\tilde{\sigma}^* \leftarrow \mathsf{A}[m^*]$. $\mathcal{B}_2$ sends $2\tilde{\sigma}^* - s^*$ to its challenger. We argue that if $\mathcal{A}$ wins its game and if $\mathcal{B}_2$ does not abort, then $\mathcal{B}_2$ wins its discrete log game. To see this, we note that, $(m^*, (R^*, s^*))$ must be a valid signature for $\mathcal{A}$ to win its game. This means that, $g^{s^*} = R^* \cdot (\mathsf{pk})^{c^*}$ where $c^* = \mathsf{H}(R^*, m^*)$. Next, if $\mathcal{B}_2$ does not abort, then the message m^* must belong to batch number i^* and $R^* = \mathsf{R}[m^*]$. This means that, R^* must be equal to $g^{2\tilde{\sigma}^*}/(h \cdot (pk)^{c^*})$, where $\tilde{\sigma}^* = \mathsf{A}[m^*]$, and $\mathsf{H}(R^*, m^*) = c^*$ was sampled $\mathcal{B}_2$ was responding to the SIGN query for $c = i^*$. Combining this with the above equation, we see that,

$$g^{s^*} = \frac{g^{2\tilde{\sigma}^*}}{h \cdot \mathsf{pk}^{c^*}} \cdot \mathsf{pk}^{c^*} = \frac{g^{2\tilde{\sigma}^*}}{h}$$

The above implies that $h = g^{2\tilde{\sigma}^* - s^*}$, meaning that $\mathcal{B}_2$ indeed computes the correct discrete log, and hence wins its game. We now analyse the probability of $\mathcal{B}_2$ not aborting. Let E_g denote the event that $\mathcal{B}_2$ correctly guessed the batch number i^* for the forgery message m^*. Since $\mathcal{B}_2$ samples i^* from $[q_s]$ uniformly at random, we have that $\Pr[\mathsf{E}_g] = 1/q_s$. Next, for every $i \in [n]$, let $\mathsf{E}_{r,i}$ be the event that $\mathcal{B}_2$ aborts when answering a SIGN query because (R_i^*, m_i) was already in the map H – in other words, $\mathcal{A}$ had queried the random oracle on these inputs. Since c_i is sampled uniformly randomly for all $i \in [n]$, R_i^* is a uniformly random element, from the view of the adversary. This means, that $\Pr[\mathsf{E}_{r,i}] \leq q_H/p$, where q_H is an upper bound on the number of random oracle queries by $\mathcal{A}$. Then, we get that,

$$\begin{aligned}\mathsf{Adv}^{\mathrm{dl}}_{\mathcal{B}_2}(\lambda) &= \Pr[\mathsf{SignerFairness}^{\mathsf{FSE},\Pi}_{\mathcal{A}}(1^\lambda) = 1 \wedge \mathsf{E}_2 \wedge \mathsf{E}_g \textstyle\bigwedge_{i\in[n]} \neg\mathsf{E}_{r,i}] \\ &\geq \Pr[\mathsf{SignerFairness}^{\mathsf{FSE},\Pi}_{\mathcal{A}}(1^\lambda) = 1 \wedge \mathsf{E}_2 \wedge \mathsf{E}_g] - \Pr[\textstyle\bigvee_{i\in[n]} \mathsf{E}_{r,i}] \\ &\geq \tfrac{1}{q_s} \Pr[\mathsf{SignerFairness}^{\mathsf{FSE},\Pi}_{\mathcal{A}}(1^\lambda) = 1 \wedge \mathsf{E}_2] - \tfrac{n\cdot q_H}{p}\end{aligned}$$

□

A.2 Client fairness

Proof of Theorem 3. Assume there are indices $i \in [c]$ and $j \in [n]$ such that $\mathcal{B}[i] = 1$ and $(m, \sigma) := \mathcal{F}[j]$ with $\Pi.\mathsf{Verify}(\mathsf{pk}, m, \sigma) = 0$. Let $(\{m_i\}_{i\in[n]}, \{\tilde{\sigma}_i\}_{i\in[n]}, \mathsf{aux}, \mathsf{com}_k) := \mathcal{I}[c]$, where according to the definition, $m_j = m$. Given that $\mathcal{B}[i] = 1$, it follows that:

$$\begin{aligned}&\mathsf{FSE.PVerify}(\{m_i\}_{i\in[n]}, \{\tilde{\sigma}_i\}_{i\in[n]}, \mathsf{aux}, \mathsf{com}_k) = 1 \\ &\iff g^{2\tilde{\sigma}_i} = \mathsf{com}_k \cdot R_i \cdot \mathsf{pk}^{c_i}\end{aligned}$$

Additionally, we know that $g^k = \mathsf{com}_k$; otherwise, $\mathcal{F}[c] = \perp$. Now, assume for the sake of contradiction that there exists some i such that $\Pi.\mathsf{Verify}(\mathsf{pk}, m_i, \sigma_i) = 0$. As shown in Figure 11, this implies that $g^{s_i} \neq R_i \cdot \mathsf{pk}^{c_i}$, where $s_i = 2\tilde{\sigma}_i - k$. However, we also have:

$$\begin{aligned}R_i \cdot \mathsf{pk}^{c_i} &= g^{2\tilde{\sigma}_i} \cdot \mathsf{com}_k^{-1} \\ &= g^{2\tilde{\sigma}_i - k} \\ &= g^{s_i}\end{aligned}$$

This directly contradicts the assumption that $\Pi.\mathsf{Verify}(\mathsf{pk}, m_i, \sigma_i) = 0$, thereby completing the proof.

B Blind **FSE**

B.1 Blind Signature Definition

Definition 7 (Blind Signature Scheme). *A blind signature scheme* BS *consists of the following algorithms:*

- $pp \leftarrow \mathsf{BS.Setup}(1^\lambda)$*: The setup algorithm takes the security parameter* λ *in unary and returns public parameters* pp*.*
- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BS}.Gen(pp)$*: The key generation algorithm takes the public parameters* pp *and returns a secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$*.*
- $(b, \sigma) \leftarrow \langle \mathsf{BS}.signer(\mathsf{sk}), \mathsf{BS}.user(\mathsf{pk}, m)\rangle$*: An interactive protocol is run between the signer, with private input a secret key* sk*, and the user, with the corresponding public key* pk *and a private message* m*. The signer outputs* $b = 1$ *if the interaction completes successfully and* $b = 0$ *otherwise, while the user outputs a signature* σ *if it terminates correctly, and* $\perp$ *otherwise. For a 2-round protocol, we can describe the interaction by the following algorithms:*

$$\begin{aligned}
(\mathsf{msg}_{user,0}, st_{user,0}) &\leftarrow \mathsf{BS}.user_0(\mathsf{pk}, m)\\
(\mathsf{msg}_{signer,1}, st_{signer}) &\leftarrow \mathsf{BS}.signer_0(\mathsf{sk}, \mathsf{msg}_{user,0})\\
(\mathsf{msg}_{user,1}, st_{user,1}) &\leftarrow \mathsf{BS}.user_1(st_{user,0}, \mathsf{msg}_{signer,1})\\
(\mathsf{msg}_{signer,2}, b) &\leftarrow \mathsf{BS}.signer_1(st_{signer}, \mathsf{msg}_{user,1})\\
\sigma &\leftarrow \mathsf{BS}.user_2(st_{user,1}, \mathsf{msg}_{signer,2})
\end{aligned}$$

Typically, $\mathsf{BS}.user_0$ *just initiates the session, and thus* $\mathsf{msg}_{user,0} = ()$ *and* $st_{user,0} = (\mathsf{pk}, m)$*.*

- $b \leftarrow \mathsf{BS}.Verify(\mathsf{pk}, m, \sigma)$*: The (deterministic) verification algorithm takes a public key* pk*, a message* m*, and a signature* σ*, and returns* 1 *if* σ *is valid on* m *under* pk*, and* 0 *otherwise.*

Definition 8 (Blindness). *We say a blind signature scheme* BS *(as defined in Definition 7) has blindness if any PPT adversary* $\mathcal{A}$ *wins the game Figure 13 game with probability* $\leq \frac{1}{2} + negl(\lambda)$*.*

```
$\mathsf{Blindness}_{\mathcal{A}}^{\mathsf{BS}}(1^\lambda)$
1: $b \leftarrow \{0,1\}$
2: $b_0 := b$; $b_1 := 1 - b$
3: $b' \leftarrow \mathcal{A}^{\mathsf{INIT}(\cdot), \mathsf{U}_1(\cdot), \mathsf{U}_2(\cdot)}(1^\lambda)$
4: return $(b' = b)$
```

```
Oracle $\mathsf{U}_1(i, R_{i,0}, R_{i,1})$
1: if $i \notin \{0, 1\} \vee \mathsf{sess}_i \neq \mathsf{init}$ then return $\perp$
2: $\mathsf{sess}_i := \mathsf{open}$
3: $(\mathsf{st}_{i,0}, c_{i,0}) \leftarrow \mathsf{BS.client}_1(\mathsf{pk}, R_{i,0}, m_{b_i})$
4: $(\mathsf{st}_{i,1}, c_{i,1}) \leftarrow \mathsf{BS.client}_1(\mathsf{pk}, R_{i,1}, m_{b_i})$
5: return $(c_{i,0}, c_{i,1})$
```

```
Oracle $\mathsf{INIT}(\mathsf{pk}, m_0, m_1)$
1: $\mathsf{sess}_0 := \mathsf{init}$
2: $\mathsf{sess}_1 := \mathsf{init}$
```

```
Oracle $\mathsf{U}_2(i, s_i, \beta_i)$
1: if $\mathsf{sess}_i \neq \mathsf{open}$ then return $\perp$
2: $\mathsf{sess}_i := \mathsf{closed}$
3: $\sigma_{b_i} \leftarrow \mathsf{BS.client}_2(\mathsf{st}_{i,\beta_i}, s_i)$
4: if $\mathsf{sess}_0 = \mathsf{sess}_i = \mathsf{closed}$ then
5:    1: if $\sigma_0 = \perp \vee \sigma_1 = \perp$ then $(\sigma_0, \sigma_1) := (\perp, \perp)$
      2: return $(\sigma_0, \sigma_1)$ else return $\epsilon$
```

Fig. 13: Blindness game for the clause blind Schnorr signature scheme

```
Game $\mathsf{SignerFairness}_{\mathcal{A}}^{\mathsf{FBSE},\,\mathsf{BS}}(1^\lambda)$
1 : $\mathsf{pp} \leftarrow \mathsf{FBSE.Setup}(1^\lambda)$
2 : $(\mathsf{pk},\, \mathsf{sk}) \leftarrow \mathsf{FBSE.KGen}(\mathsf{pp},\, 1^\lambda)$
3 : $\mathcal{S}_1 := \phi,\, \mathcal{S}_2 := \phi,\, \mathcal{K} := \phi,\, \mathcal{N} := \phi,\, c = 0,\, m = 0$
4 : $\{(m_i^*,\, \sigma_i^*)\}_{i \in m'} \leftarrow \mathcal{A}^{\mathsf{S}_1(\cdot),\, \mathsf{S}_2(\cdot),\, \mathsf{EXEC}(\cdot)}(\mathsf{pp},\, \mathsf{pk})$
5 : return $m < m' \wedge$
6 :     $\forall\, i \in [m'] \,:\, \mathsf{BS.Verify}(\mathsf{pk},\, m_i^*,\, \sigma_i^*) = 1 \wedge$
7 :     $\forall\, i \neq j \in [m'] \,:\, m_i^* \neq m_j^*$

Oracle $\mathsf{EXEC}(j, \mathsf{com}_k)$
1 : if $j \notin \mathcal{S}_2 \vee \mathsf{com}_k \neq com(\mathcal{K}[j])$ return $\perp$
2 : $k_j := \mathcal{K}[j];\, n_j := \mathcal{N}[j]$
3 : $m = m + n_j$
4 : return $k_j$

Oracle $\mathsf{S}_1(\mathsf{msg})$
1 : $c \leftarrow c + 1$
2 : $(\mathsf{msg}',\, \mathsf{st}_c^{(0)}) \leftarrow \mathsf{FBSE.signer}_1(\mathsf{sk},\, \mathsf{msg})$
3 : $n := \mathsf{st}_c^{(0)}.n;\, \mathcal{N}[c] = n$
4 : $\mathcal{S}_1 := \mathcal{S}_1 \cup \{c\}$
5 : return $(c,\, \mathsf{msg}')$

Oracle $\mathsf{S}_2(c,\, \mathsf{msg})$
1 : if $c \notin \mathcal{S}_1$ then return $\perp$
2 : $(\mathsf{msg}',\, k_c,\, b_c) \leftarrow \mathsf{FBSE.signer}_2(\mathsf{st}_c^{(0)},\, \mathsf{msg})$
3 : if $b_c = 1$ then $\mathcal{S}_1 = \mathcal{S}_1 \setminus \{c\},\, \mathcal{K}[c] = k_c,\, \mathcal{S}_2 = \mathcal{S}_2 \cup \{c\}$
4 : return $\mathsf{msg}'$
```

Fig. 15: Signer fairness game for fair blind signature exchange protocol FBSE.

Definition 9 (Strong Unforgeability). *We say a blind signature scheme* BS *(as defined in Definition 7) has strong unforgeability if any PPT adversary $\mathcal{A}$ wins Figure 1 game with probability negl(λ).*

B.2 FBSE Security Definitions

We require the FBSE for hidden messages to meet the following properties: *signer fairness*, *client fairness*, and *client privacy*. Intuitively, since the signing protocol is interactive now, it changes oracles in the definitions and that's the main difference between the definition of singer/client fairness here and for non-hidden messages. Finally, client privacy, akin to the concept of a blind signature, ensures that the signer does not learn which message it has signed through the protocol. We formalize these properties as follows.

Definition 10 (Signer Fairness). *We say the scheme FBSE has signer-fairness if any PPT adversary $\mathcal{A}$ wins the signer fairness game in Figure 15 game with negligible probability.*

Definition 11 (Client Fairness). *We say the scheme FBSE has client-fairness if any PPT adversary $\mathcal{A}$ wins the client fairness game in Figure 16 game with negligible probability.*

Definition 12 (Client Privacy). *We say the scheme FBSE has client-privacy if any PPT adversary $\mathcal{A}$ wins the client privacy game in Figure 17 game with probability $\leq \frac{1}{2} + negl(\lambda)$.*

B.3 FBSE completeness analysis

The proof is essentially the same as the proof of Theorem 1.

B.4 FBSE server fairness analysis

Signer fairness. We do not have a formal proof, but we conjecture that our scheme can satisfy signer fairness based on the One-more Discrete log (OMDL) assumption. We leave a formal proof as future work.

```
Game $\mathsf{BlindClientFairness}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda)$
  $\mathsf{pp} \leftarrow \mathsf{FBSE.Setup}(1^\lambda)$
  $(\mathsf{pk},\ st) \leftarrow \mathcal{A}(\mathsf{pp})$
  $\mathcal{B} := \phi,\ \mathcal{I} := \phi,\ \mathcal{F} := \phi,\ \mathcal{S} := \phi,\ c \leftarrow 0$
  $(\text{out},\ k) \leftarrow \mathcal{A}^{\mathsf{CL}_0(\cdot),\,\mathsf{CL}_1(\cdot),\,\mathsf{CL}_2,\,\mathsf{EXEC}}(\mathsf{pp},\ \mathsf{pk})$
  if $\exists\ i \in [c]$ s.t. $\mathcal{B}[i] = 1\ \wedge$
      $\exists\ (m,\ \sigma) \in \mathcal{F}[i] : \Pi.\mathsf{Verify}(\mathsf{pk},\ m,\ \sigma) = 0 :$
      return $1$
  else return $0$

Oracle $\mathsf{CL}_0(\mathsf{pk},\ \{m_i\}_{i\in[n]})$
  $c = c + 1$
  $\mathsf{st}^{(0)} := (\mathsf{pk},\ \{m_i\}_{i\in[n]})$
  $\mathcal{M}[c] = \{m_i\}_{i\in[n]}$

Oracle $\mathsf{CL}_1(c,\ \mathsf{msg})$
  $(\mathsf{st}^{(1)},\ \mathsf{msg}') \leftarrow \mathsf{FBSE.client}_1(\mathsf{st}_c^{(0)},\ \mathsf{msg})$
  return $\mathsf{msg}'$

Oracle $\mathsf{CL}_2(\mathsf{msg})$
  $(\{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ \mathsf{com}_k) \leftarrow \mathsf{FBSE.client}_2(\mathsf{st}^{(0)},\ \mathsf{msg})$
  $\mathcal{I}[c] = (\{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ \mathsf{com}_k)$
  $\mathcal{B}[c] \leftarrow \mathsf{FBSE.PVerify}(\mathsf{pk},\ \{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ \mathsf{com}_k)$

Oracle $\mathsf{EXEC}(j,\ k,\ \mathsf{tok})$
  if $j \notin \mathcal{S}$ return $\perp$
  $(\{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ \mathsf{com}_k) := \mathcal{I}[j]$
  $\{m_i\}_{i\in[n]} := \mathcal{M}[j]$
  $\mathsf{st}_{\mathsf{E},0} \leftarrow \mathsf{FSE.Init}()$
  $\mathsf{st}_{\mathsf{E},1} \leftarrow \mathsf{FSE.ExeClient}_0(\mathsf{com}_k, \mathsf{tok})$
  $(\mathsf{tok}_s, \mathsf{st}_{\mathsf{E},2}) \leftarrow \mathsf{FSE.ExeSigner}_0(k)$
  $\{\sigma_i\}_{i\in[n]} \leftarrow \mathsf{FSE.Recover}(\{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ k)$
  $\mathcal{S} \leftarrow \mathcal{S} \setminus \{j\}$
  if $\mathsf{tok}_s = \perp$ return $\perp$
  $\mathcal{F}[j] \leftarrow \{(m_i, \sigma_i)\}_{i\in[n]}$
  return $\{\sigma_i\}_{i\in[n]}$
```

Fig. 16: Client fairness game for fair blind signature exchange protocol FBSE

```
Game $\mathsf{ClientPrivacy}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda)$
  $pp \leftarrow \mathsf{FBSE.Setup}(1^\lambda)$
  $b \leftarrow\$ \{0,\ 1\}$
  $b_0 \leftarrow b,\ b_1 \leftarrow 1 - b$
  $b' \leftarrow \mathcal{A}^{\mathsf{Init}(\cdot),\,\mathsf{CL}_1(\cdot),\,\mathsf{CL}_2(\cdot)}(pp)$
  return $b = b'$

Oracle $\mathsf{INIT}(\mathsf{pk},\ \{m_i^{(0)}\}_{i\in[n]}, \{m_i^{(1)}\}_{i\in[n]})$
  $\mathsf{sess}_0 := \mathsf{init}$
  $\mathsf{sess}_1 := \mathsf{init}$
  $\mathsf{st}_0^{(0)} := (\mathsf{pk},\ \{m_i^{(b_0)}\}_{i\in[n]})$
  $\mathsf{st}_0^{(1)} := (\mathsf{pk},\ \{m_i^{(b_1)}\}_{i\in[n]})$

Oracle $\mathsf{CL}_1(j,\ \mathsf{msg}_j)$
  if $j \notin \{0,\ 1\} \vee \mathsf{sess}_j \neq \mathsf{init}$ then return $\perp$
  $\mathsf{sess}_j := \mathsf{open}$
  $(\mathsf{st}_1^{(j)},\ \mathsf{msg}_j') \leftarrow \mathsf{FBSE.client}_1(\mathsf{st}_0^{(j)},\ \mathsf{msg}_j)$
  return $\mathsf{msg}_j'$

Oracle $\mathsf{CL}_2(j,\ k,\ \mathsf{msg}_j)$
  if $\mathsf{sess}_j \neq \mathsf{open}$ then return $\perp$
  $(b^{(j)},\ \{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{com}_k,\ \mathsf{aux}) \leftarrow \mathsf{FBSE.client}_2(\mathsf{st}_1^{(j)},\ \mathsf{msg}_j)$
  if $b^{(j)} = \mathbf{false} \vee \mathsf{com}_k \neq g^k$ then return $\perp$
  $\mathsf{sess}_j := \mathsf{closed}$
  $\{\sigma_i^{(j)}\}_{i\in[n]} \leftarrow \mathsf{FBSE.Recover}(\{\tilde{\sigma}_i\}_{i\in[n]},\ \mathsf{aux},\ k)$
  if $\mathsf{sess}_0 = \mathsf{sess}_1 = \mathsf{closed}$ return $(\{\sigma_i^{(b_0)}\}_{i\in[n]},\ \{\sigma_i^{(b_1)}\}_{i\in[n]})$
```

Fig. 17: Client privacy game for fair blind signature exchange protocol FBSE.

B.5 FBSE client fairness analysis

Theorem 4. *Every adversary* PPT $\mathcal{A}$ *wins* $\mathsf{ClientFairness}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda)$ *with an advantage of* 0.

Proof. Assume there are indices $i \in [c]$ and $j \in [n]$ such that $\mathcal{B}[i] = 1$ and $(m, \sigma) := \mathcal{F}[j]$ with $\Pi.\mathsf{Verify}(\mathsf{pk}, m, \sigma) = 0$. Let $\{m_i\}_{i\in[n]} := \mathcal{M}[c]$ and $(\{\tilde{\sigma}_i\}_{i\in[n]}, \mathsf{aux}, \mathsf{com}_k) := \mathcal{I}[c]$, where according to the definition, $m_j = m$. Given that $\mathcal{B}[i] = 1$, it follows that:

$$\begin{aligned}\mathsf{FBSE.PVerify}(\{m_i\}_{i\in[n]}, \{\tilde{\sigma}_i\}_{i\in[n]}, \mathsf{aux}, \mathsf{com}_k) = 1 \\ \iff g^{2\tilde{\sigma}_i} = \mathsf{com}_k \cdot R_i \cdot \mathsf{pk}^{c_i}\end{aligned}$$

Additionally, we know that $g^k = \mathsf{com}_k$; otherwise, $\mathcal{F}[c] = \perp$. Now, for the sake of contradiction, assume there exists some i such that $\Pi.\mathsf{Verify}(\mathsf{pk}, m_i, \sigma_i) = 0$. As illustrated in Figure 11, this would imply $g^{s_i} \neq R_i \cdot \mathsf{pk}^{c_i}$, where $s_i = 2\tilde{\sigma}_i - k$. However, we also have:

$$R_i \cdot \mathsf{pk}^{c_i} = \mathsf{com}_i = g^{2\tilde{\sigma}_i} \cdot \mathsf{com}_k^{-1} = g^{2\tilde{\sigma}_i - k} = g^{s_i}$$

This directly contradicts the assumption that $\mathsf{BS.Verify}(\mathsf{pk}, m_i, \sigma_i) = 0$, thereby completing the proof. □

B.6 FBSE client privacy analysis

In $\mathsf{ClientPrivacy}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda)$, the adversarial signer $\mathcal{A}$ only observes the messages $\{c_{i,0}^{(0)}, c_{i,1}^{(0)}\}_{i\in[n]}, \{c_{i,0}^{(1)}, c_{i,1}^{(1)}\}_{i\in[n]}$ and the final signatures $\{\sigma_i^0\}_{i\in[n]}, \{\sigma_i^1\}_{i\in[n]}$. To analyze this, we define two modified games and argue that $\mathcal{A}$ wins each of these games with approximately the same probability, except for a negligible difference.

– $\mathsf{CP*}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda)$: This game is similar to the original, except that the Oracle $\mathsf{Client}_1(j, \mathsf{msg})$ is modified so that, for $j = 0$, it samples random elements $\{c_{i,0}^{(0)}, c_{i,1}^{(0)}\}_{i\in[n]} \leftarrow\$ \mathbb{Z}_p^{2n}$. It also samples $\sigma_{i,0}^{(0)}, \sigma_{i,1}^{(0)}$ randomly from $\mathbb{Z}_p$, for al $i \in [n]$. As in the original game, challenger samples $\beta_{i,0}^{(0)}, \beta_{i,1}^{(0)}$ uniformly randomly from $\mathbb{Z}_p$. It then computes $\{R_{i,b}^{(0)\prime}\}$ as follows:

$$R_{i,b}^{(0)\prime} \leftarrow \frac{g^{\sigma_{i,b}^{(0)}}}{\mathsf{pk}^{c_{i,b}^{(0)} - \beta_{i,b}^{(0)}}} \;\forall\; i \in [n], b \in \{0,1\}$$

Additionally, we program the random oracle to satisfy:

$$\mathsf{H}(R_{i,0}^{(0)\prime}, m_i^{(b_0)}) = c_{i,0}^{(0)} - \beta_{i,0}^{(0)} \quad \text{and} \quad \mathsf{H}(R_{i,1}^{(0)\prime}, m_i^{(b_0)}) = c_{i,1}^{(0)} - \beta_{i,1}^{(0)}$$

The challenger responds with $\{c_{i,0}^{(0)}, c_{i,1}^{(0)}\}_{i\in[n]}$ in the oracle query $\mathsf{Client}_1(0, \mathsf{msg})$. Next, to respond to the oracle query $\mathsf{Client}_2(\cdot)$, the challenger uses the signatures $\sigma_{i,b}^{(0)}$ as the output for the zeroth session, where the bit b is chosen for each $i \in [n]$ based on the adversary's choice.

– $\mathsf{CP**}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda)$: This game is defined similarly to $\mathsf{CP*}$, but the Oracle $\mathsf{Client}_1(j, \mathsf{msg})$ now returns random elements $\{c_{i,0}^{(j)}, c_{i,1}^{(j)}\}_{i\in[n]}$ for both $j = 0$ and $j = 1$. Additionally, we sample signatures $\sigma_{i,b}^{(1)}$ and betas $\beta_{i,b}^{(1)}$ uniformly randomly from $\mathbb{Z}_p$, for all $i \in [n], b \in \{0,1\}$, similar to the previous game. We compute $\{R_{i,b}^{(1)'}\}$ as follows:

$$R_{i,b}^{(1)'} \leftarrow \frac{g^{\sigma_{i,b}^{(1)}}}{\mathsf{pk}^{c_{i,b}^{(1)} - \beta_{i,b}^{(1)}}}$$

. Lastly, we program the random oracle also to satisfy:

$$\mathsf{H}(R_{i,0}^{(1)\prime}, m_i^{(b_1)}) = c_{i,0}^{(1)} - \beta_{i,0}^{(1)} \quad \text{and} \quad \mathsf{H}(R_{i,1}^{(1)\prime}, m_i^{(b_1)}) = c_{i,1}^{(1)} - \beta_{i,1}^{(1)}$$

In this game, the challenger uses the uniformly sampled $c_{i,b}^{(j)}$ values to respond to the Client_1 queries, and uses the uniformly sampled $\sigma_{i,b}^{(j)}$ values to respond to the Client_2 oracle queries.

Next, we define two following events:

- E_1: In $\mathsf{CP*}$, E_1 is the event that $\mathcal{A}$ queries either $\mathsf{H}(R_{i,0}^{(0)\prime}, m_i^{(b_0)})$ or $\mathsf{H}(R_{i,1}^{(0)\prime}, m_i^{(b_0)})$ before the challenger computes $R_{i,0}^{(0)\prime}$ and $R_{i,1}^{(0)\prime}$.
- E_2: In $\mathsf{CP**}$, E_2 is the event that $\mathcal{A}$ queries any of $\mathsf{H}(R_{i,0}^{(0)\prime}, m_i^{(b_0)})$, $\mathsf{H}(R_{i,1}^{(0)\prime}, m_i^{(b_0)})$, $\mathsf{H}(R_{i,0}^{(1)\prime}, m_i^{(b_1)})$, or $\mathsf{H}(R_{i,1}^{(1)\prime}, m_i^{(b_1)})$ before the challenger computes the respective values.

We assert that both events E_1 and E_2 occur with negligible probability. It simply follows from the fact that values R values are uniform. Now, we compare the advantage of $\mathcal{A}$ in the original game with the modified games. First, consider:

$$\begin{aligned}\Pr[\mathsf{ClientPrivacy}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda) = 1] &\leq \Pr[\mathsf{ClientPrivacy}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda) = 1 \,|\, \overline{E}_1] + \Pr[E_1]\\ &= \Pr[\mathsf{CP*}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda) = 1] + \Pr[E_1]\end{aligned}$$

Since $\Pr[E_1]$ is negligible in λ, we conclude that the probability of $\mathcal{A}$ winning in $\mathsf{ClientPrivacy}$ and $\mathsf{CP*}$ differs by a negligible factor. By similar reasoning, we have:

$$\begin{aligned}\Pr[\mathsf{CP*}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda) = 1] &\leq \Pr[\mathsf{CP*}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda) = 1 \,|\, \overline{E}_2] + \Pr[E_2]\\ &= \Pr[\mathsf{CP**}_{\mathcal{A}}^{\mathsf{FBSE,\,BS}}(1^\lambda) = 1] + \Pr[E_2]\end{aligned}$$

Since $\Pr[E_2]$ is negligible in λ, we conclude that the probability of $\mathcal{A}$ winning in $\mathsf{CP*}$ and $\mathsf{CP**}$ also differs by a negligible factor. Finally, in the game $\mathsf{CP**}$, the adversary only receives randomly selected elements $\{c_{i,0}^{(0)}, c_{i,1}^{(0)}\}_{i\in[n]}$, $\{c_{i,0}^{(1)}, c_{i,1}^{(1)}\}_{i\in[n]}$ along with randomly sampled signatures $\{\sigma_i^{(b_0)}\}$, $\{\sigma_i^{(b_1)}\}$. This implies that $\mathcal{A}$ can win this game with a probability of at most $\frac{1}{2}$. Therefore, we conclude that $\mathcal{A}$ wins CP with a probability that is negligibly different from $\frac{1}{2}$.