Foundations of Coin Mixing Services

Noemi Glaeser  
University of Maryland  
College Park, MD, USA  
Max Planck Institute for Security and Privacy  
Bochum, Germany  
goinglaeser@umd.edu

Matteo Maffei  
TU Wien  
Christian Doppler Laboratory  
Blockchain Technologies for the Internet of Things  
Vienna, Austria  
matteo.maffei@tuwien.ac.at

Giulio Malavolta  
Max Planck Institute for Security and Privacy  
Bochum, Germany  
giulio.malavolta@hotmail.it

Pedro Moreno-Sanchez  
IMDEA Software Institute  
Madrid, Spain  
pedro.moreno@imdea.org

Erkan Tairi  
TU Wien  
Christian Doppler Laboratory  
Blockchain Technologies for the Internet of Things  
Vienna, Austria  
erkan.tairi@tuwien.ac.at

Sri AravindaKrishnan  
Thyagarajan  
Carnegie Mellon University  
Pittsburgh, PA, USA  
t.srikrishnan@gmail.com

ABSTRACT
Coin mixing services allow users to mix their cryptocurrency coins and thus enable unlinkable payments in a way that prevents tracking of honest users’ coins by both the service provider and the users themselves. The easy bootstrapping of new users and backwards compatibility with cryptocurrencies (such as Bitcoin) with limited support for scripts are attractive features of this architecture, which has recently gained considerable attention in both academia and industry.

A recent work of Tairi et al. [IEEE S&P 2021] formalizes the notion of a coin mixing service and proposes A²L, a new cryptographic protocol that simultaneously achieves high efficiency and interoperability. In this work, we identify a gap in their formal model and substantiate the issue by showing two concrete counterexamples: we show how to construct two encryption schemes that satisfy their definitions but lead to a completely insecure system.

To amend this situation, we investigate secure constructions of coin mixing services. First, we develop the notion of blind conditional signatures (BCS), which acts as the cryptographic core for coin mixing services. We propose game-based security definitions for BCS and propose A²L⁺, a modified version of the protocol by Tairi et al. that satisfies our security definitions. Our analysis is in an idealized model (akin to the algebraic group model) and assumes the hardness of the one-more discrete logarithm problem. Finally, we propose A²L¹UC, another construction of BCS that achieves the stronger notion of UC-security (in the standard model), albeit with a significant increase in computation cost. This suggests that constructing a coin mixing service protocol secure under composition requires more complex cryptographic machinery than initially thought.

1 INTRODUCTION
Bitcoin and cryptocurrencies sharing Bitcoin’s core principles have attained huge prominence as decentralized and publicly verifiable payment systems. They have attracted not only cryptocurrency enthusiasts but also banks [5], leading IT companies (e.g., Facebook and PayPal), and payment providers such as Visa [20]. At the same time, the initial perception of payment unlinkability based on pseudonyms has been refuted in numerous academic research works [39, 51], and the blockchain surveillance industry [30] demonstrates this privacy breach in practice. This has led to a large amount of work devoted to providing a privacy-preserving overlay to Bitcoin in the form of coin mixing protocols [6, 26].

Decentralized coin mixing protocols such as CoinJoin [1] or CoinShuffle [45–47] allow a set of mutually distrusting users to mix their coins to achieve unlinkability: that is, the coins cannot be linked to their initial owners even by malicious participants. These protocols suffer from a common drawback, the bootstrapping problem, i.e., how to find a set of participants to execute the protocol. In fact, while a high number of participants is desirable to improve the anonymity guarantees provided by the coin mixing protocol, such a high number is at the same time undesirable as it results in poor scalability and makes bootstrapping harder.

An alternative mechanism is one in which a third party, referred to as the hub, alleviates the bootstrapping problem by connecting users that want to mix their coins. Moreover, the hub itself can provide a coin mixing service by acting as a tumbler. In more detail, users send their coins to the hub, which, after collecting all the coins, sends them back to the users in a randomized order, thereby providing unlinkability for an observer of such transfers (e.g., an observer of the corresponding Bitcoin transactions).

Synchronization Puzzles. There are numerous reported cases of “exit scams” by mixing services which took in new payments but stopped providing the mixing service [52]. This has prompted
the design of numerous cryptographic protocols [2, 14, 32, 54] to remove trust from the hub, providing a trade-off between trust assumptions, minimum number of transactions, and Bitcoin compatibility [31]. Of particular interest is the work by Heilman et al. [31], which lays the groundwork for the core cryptographic primitive which can be used to build a mixing service. This primitive, referred to as a synchronization puzzle, enables unlinkability from even the view of a corrupt hub. However, Heilman et al. only present informal descriptions of the security and privacy notions of interest. Furthermore, the protocol proposed (TumbleBit) relies on hashed time-lock contracts (HTLCs), a smart contract incompatible with major cryptocurrencies such as Monero, Stellar, Ripple, MimbleWimble, and Zerocash (shielded addresses), lowering the interoperability of the solution.

The recent work of Tairi et al. [53] attempts to overcome both of these limitations. It gives formal security notions for a synchronization puzzle in the universal composability (UC) framework [16]. It also provides an instantiation of the synchronization puzzle (called $A^2L$) that is simultaneously more efficient and more interoperable than TumbleBit, requiring only timelocks and digital signature verification from the underlying cryptocurrencies.

In this work, we identify a gap in their security analysis, and we substantiate the issue by presenting two concrete counterexamples: there exist two encryption schemes (secure under standard cryptographic assumptions) that satisfy the prerequisites of their security notions, yet yield completely insecure systems. This shows that our understanding of synchronization puzzles as a cryptographic primitive is still inadequate. Establishing firm foundations for this important cryptographic primitive requires us to rethink this object from the ground up.

1.1 Our Contributions

We summarize the contributions of this work below.

Counterexamples. First, we identify a gap in the security model of the synchronization puzzle protocol $A^2L$ [53], presenting two concrete counterexamples (Section 3). Specifically, we show that there exist underlying cryptographic building blocks that satisfy the prerequisites stated in $A^2L$, yet they allow for:

- a key recovery attack, in which a user can learn the long-term secret decryption key of the hub;
- a one-more signature attack, in which a user can obtain $n$ signed transactions from the hub while only engaging in $n-1$ successful instances of signing a transaction which pays the hub. In other words, the user obtains $n$ coins from the hub while the hub receives only $n-1$ coins.

Both attacks run in polynomial time and succeed with overwhelming probability.

Definitions. To place the synchronization puzzle on firmer foundations, we propose a new cryptographic notion that we call blind conditional signatures (BCS). Our new notion intuitively captures the functionality of a synchronization puzzle from [31, 53]. BCS is a simple and easy-to-understand tool, and we formalize its security notions both in the game-based (Section 4.1) and universal composability (Section 5) setting. The proposed game-based definitions for BCS are akin to the well-understood standard security notions for regular blind signatures [19, 48]. We hope that this abstraction may lay the foundations for further studies on this primitive in all cryptocurrencies, scriptless or not.

Constructions. We give two constructions, one that satisfies our game-based security guarantees and one that is UC-secure. Both require only the same limited functionality as $A^2L$ from the underlying blockchain. In more detail:

- We give a modified version of $A^2L$ (Sections 4.2 and 4.3) which we refer to as $A^2L^+$ that satisfies the game-based notions (Section 4.1) of BCS, albeit in the linear-only encryption (LOE) model [29]. In this model, the attacker does not directly have access to a homomorphic encryption scheme; instead, it can perform the legal operations by querying the corresponding oracles. This is a strong model with a non-falsifiable flavor, similar to the generic/algebraic group model [24, 37, 50].
- We then provide a less efficient construction $A^2L^{UC}$ that securely realizes the UC notion of BCS (Section 5). This scheme significantly departs from the construction paradigm of $A^2L$ and is based on general-purpose cryptographic tools such as secure two-party computation (2PC).

Our results hint at the fact that achieving UC-security for a synchronization puzzle requires a radical departure from current construction paradigms, and it is likely to lead to less efficient schemes. On the other hand, we view the game-based definitions (a central contribution of our work) as a reasonable middle ground between security and efficiency.

1.2 Technical Overview

To put our work into context, we give a brief overview of $A^2L$ [53] recast as a synchronization puzzle (a notion first introduced in [31]), and discuss how it can be used as a coin mixing protocol. We then outline the vulnerabilities in $A^2L$ and discuss how to fix them using the tools that we develop in this work.

Synchronization Puzzles. A synchronization puzzle protocol is a protocol between three parties: Alice, Bob, and Hub (refer to Figure 1 for a pictorial description). The synchronization puzzle begins with Hub and Bob executing a puzzle promise protocol (step 1) with respect to some message, $m_{HB}$ such that Bob receives a puzzle $\tau$ that contains a signature $s$ (at this point still hidden) on $m_{HB}$. Bob wishes to solve the puzzle and obtain the embedded signature. To do this, he sends the puzzle $\tau$ privately to Alice (step 2), who executes a puzzle solve protocol (step 3) with Hub with respect to some message $m_{AH}$ such that, at the end of the protocol, Alice obtains the signature $s$, whereas Hub obtains a signature $s'$ on $m_{AH}$. Alice then sends the signature $s$ privately to Bob (step 4). Such a protocol must satisfy the following properties.

Blindness: The puzzle solve protocol does not leak any information to Hub about $\tau$, and Hub blindly helps solve the puzzle. This ensures that Hub cannot link puzzles across interactions.

Unlockability: If step 3 is successfully completed, then the secret $s$ must be a valid secret for Bob’s puzzle $\tau$. This guarantees that Hub cannot learn a signature on $m_{AH}$, without at the same time revealing a signature on $m_{HB}$.

Unforgeability: Bob cannot output a valid signature on $m_{HB}$ before Alice interacts with the Hub.
Towards a Coin Mixing Service. As shown in [31, 53], the synchronization puzzle is the cryptographic core of a coin mixing service. First, Alice and Bob define the messages

\[ m_{AH} : (A \xrightarrow{v} H) \text{ and } m_{HB} : (H \xrightarrow{v} B) \]

where \((U_i \xrightarrow{v} U_j)\) denotes a cryptocurrency payment (e.g., on-chain transaction or a payment over payment channels) that transfers \(v\) coins from \(U_i\) to \(U_j\). Second, Alice and Bob run the synchronization puzzle protocol with Hub to synchronize the two aforementioned transfers. Here, the signatures \(s\) and \(s'\) are the ones required to validate the transactions defined by \(m_{AH}\) and \(m_{HB}\). The anonymity of mixing follows from the fact that multiple pairs of users are executing the synchronization puzzle simultaneously with Hub, and Hub cannot link its interaction on the left to the corresponding interaction on the right. Throughout the rest of this work, we mainly focus on the synchronization puzzle as a cryptographic primitive. The application of a coin mixing protocol follows as prescribed in prior works [31, 53].

The \(A^2L\) System. In \(A^2L\), the blindness property is achieved by making use of a re-randomizable linearly homomorphic (CPA-secure) encryption. The puzzle \(r\) contains a ciphertext \(c \leftarrow \text{Enc}(ek_H, s)\) encrypting the signature \(s\) under the encryption key \(ek_H\) of Hub. During the puzzle solve step, Alice first re-randomizes the ciphertext (and the underlying plaintext)

\[ c \xrightarrow{r} c' = \text{Enc}(ek_H, s + r) \]

with a random scalar \(r\). Hub then decrypts \(c'\) to obtain \(s + r\), which in turn reveals a signature \(s'\) on \(m_{AH}\). Alice can then strip off the re-randomization factor \(r\) and send \(s\) to Bob later in step 4.

In the analysis, it is argued that the CPA-security of the encryption scheme ensures unforgeability, whereas the re-randomization process guarantees blindness. Unfortunately, we show in this work that this claim is flawed.

Counterexamples. We observe that the encryption scheme is only CPA-secure, and the Hub is offering a decryption oracle in disguise. In these settings, the right notion of security is the stronger CCA-security, which accounts exactly for this scenario. However, CCA-security is at odds with blindness, since we require the scheme to be (i) linearly homomorphic and (ii) publicly re-randomizable. We then substantiate this concern by showing two counterexamples. Specifically, we show that there exist two encryption schemes that satisfy the prerequisites spelled out by \(A^2L\), but enable two concrete attacks against the protocol. Depending on the scheme, we can launch one of the following attacks:

- A key recovery attack that completely recovers the long-term secret key of the hub, i.e., the decryption key \(dk_H\).
- A one-more signature attack that allows one to obtain \(n + 1\) signatures on transactions from Hub to Bob, while only revealing \(n\) signatures on transactions from Alice to Hub. Effectively, this allows one to steal coins from the hub.

We stress that both these schemes are specifically crafted to make the protocol fail: their purpose is to highlight a gap in the security model of \(A^2L\). As such, they do not imply that \(A^2L\) as implemented is insecure, although we cannot prove it secure either. For a detailed description of the attacks, we refer the reader to Section 3.2.

Can We Fix This? In light of our attacks, the natural question is whether we can establish formally rigorous security guarantees for the (appropriately patched) \(A^2L\) system. While it seems unlikely that \(A^2L\) can achieve UC-security (more discussion on this later), we investigate whether it satisfies some weaker, but still meaningful, notion of security. Our main observation here is that a weak notion of CCA-security for encryption schemes suffices to provide formal guarantees for \(A^2L\). This notion, which we refer to as one-more CCA-security, (roughly) states that it is hard to recover the plaintexts of \(n\) ciphertexts while querying a decryption oracle at most \(n - 1\) times. Importantly, this notion is, in principle, not in conflict with the homomorphism/re-randomization requirements, contrary to standard CCA-security.

Towards establishing a formal analysis of \(A^2L\), we introduce the notion of blind conditional signatures (BCS) as the cryptographic cornerstone of a synchronization puzzle. We propose game-based definitions (Section 4.1) similar in spirit to the well-established security definitions of regular blind signatures [19, 48]. We then prove that \(A^2L^+\), our appropriately modified version of \(A^2L\), satisfies these definitions (Section 4.2). Our analysis comes with an important caveat: we analyze the security of our scheme in the linear-only encryption model. This is a model introduced by Groth [29] that only models adversaries that are restricted to perform “legal” operations on ciphertexts, similarly to the generic/algebraic group

\(^1\)This is achieved via the notion of adaptor signatures, but for the sake of this overview we ignore the exact details of this aspect.

\(^2\)It is well known that no encryption scheme that satisfies either of these properties can be CCA-secure.
model. While this is far from a complete analysis, it increases our confidence in the security of the system.  

**UC-Security.** The next question that we set out to answer is whether we can construct a synchronization puzzle that satisfies the strong notion of UC-security. We do not know how to prove that \( A^L \) (or \( A^{L^*} \)) is secure under composition, which is why we prove \( A^{L^*} \)-secure only in the game-based setting. The technical difficulty in proving UC-security is that blindness is unconditional, and we lack a “trapdoor mechanism” that allows the simulator to link adversarial sessions during simulation in the security analysis; the proof of UC-security in [53] is flawed due to this same reason. Thus, in Section 5.2 we develop a different protocol (called \( A^2L^{UC} \)) that we can prove UC-secure in the standard model. The scheme relies on standard general-purpose cryptographic tools, such as 2PC, and incurs a significant increase in computation costs. We stress that we view this scheme as a proof-of-concept, and leave further improvements for practical efficiency as an open problem. We hope that the scheme will shed some light on the barriers that need to be overcome in order to construct a practically efficient UC-secure synchronization puzzle.

### 1.3 Related Work

We recall some relevant related work in the literature.

**Unlinkable Transactions.** CoinJoin [1], Coinshuffle [45–47], and Möbius [38] are coin mixing protocols that rely on interested users coming together and making an on-chain transactions to mix their coins. These proposals suffer from the bootstrapping problem (users having to find other interested users for the mix) in addition to requiring custom scripting language support from the underlying currency and completing the mix with on-chain transactions. Pe-run [23] and mixEth [49] are mixing solutions that rely on Ethereum smart contracts to resolve contentions among users. An alternate design choice is to incorporate coin unlinkability natively in the currency. Monero [35] and Zcash [10] are the two most popular examples of currencies that allow for unlinkable transactions without any special coin mixing protocol. This is enabled by complex on-chain cryptographic mechanisms that are not supported in other currencies.

**RCCA Security.** A security notion related to one-more CCA is that of re-randomizable Replayable CCA (RCCA) encryption scheme [43]. The notion guarantees security even if the adversary has access to a decryption oracle, but only for ciphertexts that do not decrypt to the challenge messages. This is slightly different from what we require in our setting, since in our application the adversary will always query the oracle on encryption of new (non-challenge) messages (because of the plaintext re-randomization). This makes it challenging to leverage the guarantees provided by this notion in our analysis.

---

2 **PRELIMINARIES**

We denote by \( n \in \mathbb{N} \) the security parameter and by \( x \leftarrow \mathcal{A}(in;r) \) the output of the algorithm \( \mathcal{A} \) on input in using \( r \leftarrow \{0,1\}^* \) as its randomness. We often omit this randomness and only mention it explicitly when required. We say that an algorithm is (non-uniform) PPT if it runs in probabilistic polynomial time. We say that a function is negligible if it vanishes faster than any polynomial.

**Digital Signature.** A digital signature scheme \( \Pi_{DS} \) := (KGen, Sign, Vf) has a key generation algorithm \( (vk, sk) \leftarrow \text{KGen}(1^n) \) that outputs a verification-signing key pair. The owner of the signing key sk can compute signatures on a message \( m \) by running \( \sigma \leftarrow \text{Sign}(sk, m) \), which can be publicly verified using the corresponding verification key vk by running \( \text{Vf}(vk, m, \sigma) \). We require that the digital signature scheme satisfies the standard notion of strong existential unforgeability [28].

**Hard Relations.** We recall the notion of a hard relation \( R \) with statement/witness pairs \((Y, y)\). We denote by \( L_R \) the associated language defined as \( L_R := \{ Y \mid \exists y, (Y, y) \in R \} \). The relation is called a hard relation if the following holds: (i) There exists a PPT sampling algorithm \( \text{GenR}(1^n) \) that outputs a statement/witness pair \((Y, y) \in R \); (ii) The relation is poly-time decidable; (iii) For all \( R \) PPT adversaries \( \mathcal{A} \) the probability of \( \mathcal{A} \) on input \( Y \) outputting a witness \( y \) is negligible. In this work we use the discrete log language \( L_{DL} \) defined with respect to a group \( G \) with generator \( g \) and order \( p \). The language is defined as \( L_{DL} := \{ Y \mid \exists y \in \mathbb{Z}_p, Y = g^y \} \) with corresponding hard relation \( R_{DL} \).

**Adaptor Signatures.** Adaptor signatures [3] let users generate a pre-signature on a message \( m \) which by itself is not a valid signature, but can later be adapted into a valid signature using knowledge of some secret value. More precisely, an adaptor signature scheme \( \Pi_{ADP} := (KGen, \text{PreSig}, \text{PreVf}, \text{Adapt}, Vf, \text{Ext}) \) is defined with respect to a signature scheme \( \Pi_{DS} \) and a hard relation \( R \). The key generation algorithm is the same as in \( \Pi_{DS} \) and outputs a key pair \((vk, sk)\). The pre-signing algorithm \( \text{PreSig}(sk, m, \sigma) \) returns a pre-signature \( \sigma \) (we sometimes also refer to this as a partial signature). The pre-signature verification algorithm \( \text{PreVf}(vk, m, \sigma, y) \) verifies if the pre-signature \( \sigma \) is correctly generated. The adapt algorithm \( \text{Adapt}(\sigma, y) \) transforms a pre-signature \( \sigma \) into a valid signature \( \sigma' \) given the witness \( y \) for the instance \( Y \) of the language \( L_R \). The verification algorithm \( Vf \) is the same as in \( \Pi_{DS} \). Finally, we have the extract algorithm \( \text{Ext}(\sigma, \sigma, Y) \) which, given a pre-signature \( \sigma \), a signature \( \sigma' \), and an instance \( Y \), outputs the witness \( y \) for \( Y \). This can be formalized as pre-signature correctness.

**Definition 2.1 (Pre-signature Correctness).** An adaptor signature scheme \( \Pi_{ADP} \) satisfies pre-signature correctness if for every \( n \in \mathbb{N} \), every message \( m \in \{0,1\}^* \), and every statement/witness pair \((Y, y) \in R \), the following holds:

\[
\begin{align*}
\text{Pr} \left[ \begin{array}{l}
\text{PreVf}(vk, m, \sigma, y) = 1 \\
\text{Vf}(vk, m, \sigma, y') = 1
\end{array} \right] &= 1.
\end{align*}
\]

In terms of security, we want standard unforgeability even when the adversary is given access to pre-signatures with respect to the signing key sk.

---

3 We resort to the LOE model because of the seemingly inherent conflict between linear homomorphism and CCA-like security, both of which are needed for our application (in our setting, the adversary has access to something akin to a decryption oracle). Indeed, even proving that ElGamal encryption is CCA1-secure in the standard model is a long-standing open problem, and we believe that the \( A^L \) approach would inherently hit this barrier without some additional assumption.
Figure 2: Unforgeability experiment of adaptor signatures

**Definition 2.2 (Unforgeability).** An adaptor signature scheme $\Pi_{ADP}$ is eUF-CMA secure if for every PPT adversary $A$ there exists a negligible function $\negl$ such that

\[
\Pr\left[\text{aSigForge}_{\Pi_{ADP}}(n) = 1\right] \leq \negl(n),
\]

where the experiment $\text{aSigForge}_{\Pi_{ADP}}$ is defined as in Figure 2.

We also require that, given a pre-signature and a witness for the instance, one can always adapt the pre-signature into a valid signature (pre-signature adaptability).

Definition 2.3 (Pre-signature Adaptability). An adaptor signature scheme $\Pi_{ADP}$ satisfies pre-signature adaptability if for any $n \in \mathbb{N}$, any message $m \in \{0,1\}^n$, any statement/witness pair $(Y, y) \in R$, any key pair $(sk, vk) \leftarrow \text{KGen}(1^n)$, and any pre-signature $\hat{\sigma} \leftarrow \text{PreSig}(sk, m, Y)$, $\hat{\sigma}$ is witness extractable if for every $\ell = \text{GenR}(1^n)$, such that

\[
\Pr\left[\text{Vf}(vk, m, \hat{\sigma}) \neq \text{Adapt}(\hat{\sigma}, y) \right] = 1.
\]

Finally, we require that, given a valid pre-signature and a signature with respect to the same instance, one can efficiently extract the corresponding witness (witness extractability).

**Definition 2.4 (Witness Extractability).** An adaptor signature scheme $\Pi_{ADP}$ is witness extractable if for every PPT adversary $A$, there exists a negligible function $\negl$ such that

\[
\Pr\left[\text{aWitExt}_{\Pi_{ADP}}(n) = 1\right] \leq \negl(n),
\]

where the experiment $\text{aWitExt}_{\Pi_{ADP}}$ is defined as in Figure 3.

Figure 3: Witness extractability experiment for adaptor signatures

Combining the three properties described above, we can define a secure adaptor signature scheme as follows.

**Definition 2.5 (Secure Adaptor Signature Scheme).** An adaptor signature scheme $\Pi_{ADP}$ is secure if it is aUF-CMA secure, pre-signature adaptable, and witness extractable.

**Linear-Only Homomorphic Encryption.** A public-key encryption scheme $\Pi_E := (\text{KGen}, \text{Enc}, \text{Dec})$ allows one to generate a key pair $(ek, dk) \leftarrow \text{KGen}(1^n)$ that allows anyone to encrypt messages as $c \leftarrow \text{Enc}(ek, m)$ and allows only the owner of the decryption key $dk$ to decrypt ciphertexts as $m \leftarrow \text{Dec}(dk, c)$. We require that $\Pi_E$ satisfies perfect correctness and the standard notion of CPA-security [27]. We say that an encryption scheme is *linearly homomorphic* if there exists some efficiently computable operation $\circ$ such that $\text{Enc}(ek, m_0) \circ \text{Enc}(ek, m_1) \in \text{Enc}(ek, m_0 + m_1)$, where addition is defined over $\mathbb{Z}_p$. The $\alpha$-fold application of $\circ$ is denoted by $\text{Enc}(ek, m)^\alpha$.

Linear-only encryption (LOE) is an idealized model introduced by Groth [29] as “generic homomorphic cryptosystem”. Here, homomorphic encryption is modeled by giving access to oracles instead of their corresponding algorithms. A formal description of the oracles is given in Figure 4. We note that although we do not model such an algorithm explicitly, this model allows for (perfect) ciphertext re-randomization by homomorphically adding 0 to the desired ciphertext.

**Non-Interactive Zero-Knowledge.** Let $\mathcal{L} := \{0,1\}^* \rightarrow \{0,1\}$ be an NP-witness-relation with corresponding NP-language $L := \{x \mid \exists w.s.t. \mathcal{L}(x, w) = 1\}$. A non-active zero-knowledge proof system $NIZK := (\text{Setup}, \text{P}, \text{V})$ for the relation $\mathcal{L}$ is initialized with a setup algorithm $\text{Setup}(1^n)$ that, on input the security parameter, outputs a common reference string $\text{crs}$ and a trapdoor $td$. A prover can show the validity of a statement $x$ with a witness $w$ by invoking $\text{P}(\text{crs}, x, w)$, which outputs a proof $\pi$. The proof $\pi$ can be efficiently checked by the verification algorithm $\text{V}(\text{crs}, x, \pi)$. We require a NIZK system to be (1) zero-knowledge, i.e., there exists...
a simulator \( \pi \leftarrow \text{Sim}(\mathsf{td}, x) \) that computes valid proofs without the knowledge of the witness, (2) sound, i.e., it is infeasible for an adversary to output a valid proof for a statement \( x \notin \mathcal{L} \), and (3) UC-secure, i.e., one can efficiently extract from the proofs computed by the adversary a valid witness (with the knowledge of the trapdoor \( \mathsf{td} \)), even in the presence of simulated proofs. For formal security definitions, we refer the reader to [15, 22].

One-More DL. We recall the one-more discrete logarithm (OMDL) assumption [7, 9].

Definition 2.6 (One-More Discrete Logarithm (OMDL) Assumption). Let \( \mathcal{G} \) be a uniformly sampled cyclic group of prime order \( p \) and \( g \) a random generator of \( \mathcal{G} \). The one-more discrete logarithm (OMDL) assumption states that for all \( n \in \mathbb{N} \) there exists a negligible function \( \text{negl}(n) \) such that for all PPT adversaries \( \mathcal{A} \) making at most \( g = \text{poly}(n) \) queries to \( \text{DL}(\cdot) \), the following holds:

\[
\Pr \left[ \forall i : x_i = r_i \quad \frac{r_1, \ldots, r_{q+1} \leftarrow \text{SR}_{\mathcal{G}}}{\forall i \in [q+1], h_i \leftarrow g^{r_i} \quad \{x_i\}_{i \in [q+1]} \leftarrow \mathcal{A}_{\text{DL}(\cdot)}(h_1, \ldots, h_{q+1})} \right] \leq \text{negl}(n),
\]

where the \( \text{DL}(\cdot) \) oracle takes as input an element \( h \in \mathcal{G} \) and returns \( x \) such that \( h = g^x \).

3 COUNTEREXAMPLES OF A²L

In the following, we recall the A²L system and present two counterexamples to their main theorem.

3.1 Description of A²L

A²L is defined over the following cryptographic schemes:

- A digital signature scheme \( \PiDS \), a hard relation \( \text{RDL} \) for a group \((\mathcal{G}, g, p)\) with generator \( g \) and prime order \( p \), and the corresponding adaptor signature scheme \( \Pi_{\text{ADP}} \).
- A linearly homomorphic re-randomizable CPA-secure encryption scheme \( \Pi_E \).
- A NIZK proof system \( \Pi_{\text{NIZK}} := (\text{Setup}, \mathcal{P}, \mathcal{V}) \) for the language \( \mathcal{L} := \{ (\mathsf{ek}, y, c) \mid 3 \text{ s.t. } c = \Pi_E.\text{Enc}(\mathsf{ek}, s) \land Y = g^c \} \).

The protocol has three parties: Alice, Bob, and Hub. At the beginning of the system, Hub runs the setup (as described in Figure 9) to generate its keys, which are the keys for the (CPA-secure) encryption scheme \( \Pi_E \). The protocol then consists of a promise phase and a solving phase.

Puzzle Promise. In the promise phase (Figure 12), Hub generates a pre-signature \( \sigma_H^P \) on a common message \( m_{\mathsf{HB}} \) with respect to a uniformly sampled instance \( Y := g^x \). Hub also encrypts the witness \( s \) in the ciphertext \( c \leftarrow \Pi_E.\text{Enc}(\mathsf{ek}_H, s) \) under its own encryption key \( \mathsf{ek}_H \). Hub gives Bob the tuple \((Y, c, \pi, \sigma_H^P)\), where \( \pi \) is a NIZK proof that certifies the ciphertext \( c \) encrypts \( s \). Bob verifies that the NIZK proof and the pre-signature are indeed valid. If so, he chooses a random \( r \leftarrow \text{SR}_{\mathcal{G}} \) and re-randomizes the instance \( Y \) to \( Y' := Y \cdot g^r \) and also re-randomizes the ciphertext \( c \leftarrow \Pi_E.\text{Rand}(c, r) \). The puzzle is set to \( \tau := (r, m_{\mathsf{HB}}, \sigma_H^P, (Y, c, (Y', c')) \).

Puzzle Solve. Bob sends the puzzle \( \tau \) privately to Alice, who now executes the puzzle solve protocol with Hub (Figure 13). Alice samples a random \( r' \) and further re-randomizes the instance \( Y' \) as \( Y'' := Y' \cdot g^{r'} \) and the ciphertext \( c' \) as \( c'' \leftarrow \Pi_E.\text{Rand}(c', r') \). She then generates a pre-signature \( \sigma_{AH} \) on a common message \( m_{AH} \) with respect to the instance \( Y'' \). She sends the tuple \((Y'', c'', \sigma_{AH})\) to Hub, who decrypts \( c'' \) using the decryption key \( \mathsf{dk}_H \) to obtain \( s'' \). Hub then adapts the pre-signature \( \sigma_{AH} \) to \( \sigma_H \) using \( s'' \) and ensures its validity. It then sends the signature \( \sigma_H \) to Alice, who extracts the witness for \( Y'' \) as \( s'' \leftarrow \Pi_{\text{ADP}.\text{Ext}}(\sigma_{AH}, \sigma_H, Y'') \). Alice removes the re-randomization factor to obtain the solution \( s' := s'' - r' \) for the instance \( Y' \). Alice finally sends \( s' \) privately to Bob, who opens the puzzle \( r \) by computing the witness \( s := s' - r \) and adapting the pre-signature \( \sigma_H^P \) (given by Hub in the promise phase) to the signature \( \sigma_H^P \).

3.2 Counterexamples

Next, we describe two cryptographic instantiations of A²L that satisfy the formal definitions, yet enable two attacks. For the purpose of these attacks, it suffices to keep in mind that Hub offers the sender party (Alice) access to the following oracle, which we refer to as \( \Pi_{O^{\mathcal{L}}, \mathsf{HB}, \mathsf{ds}, \mathsf{HL}, \mathsf{I}, \mathsf{ADP}} \). On input a verification key \( v_k \), a message \( m \), a group element \( h \), a ciphertext \( c \), and a partial signature \( \tilde{s} \), the oracle behaves as follows:

- Compute \( \tilde{x} := \Pi_E.\text{Dec}(\mathsf{dk}, c) \).
- Compute \( \sigma' := \Pi_{\text{ADP}.\text{Adapt}}(\tilde{s}, \tilde{x}) \).
- If \( \Pi_{\text{ADP}.\text{Verf}}(v_k, m, \sigma') = 1 \), return \( \sigma' \).
- Else return \( \bot \).

Note that returning \( \sigma' \) implicitly reveals \( \tilde{x} \), since \( \Pi_{\text{ADP}.\text{Ext}}(\tilde{s}, \sigma', h) = \tilde{x} \). It is also useful to observe that providing a valid pre-signature to the A²L oracle is trivial for an adversary: generating a pre-signature that is valid when adapted with a value \( x \) requires only knowledge of the party’s own signing key and of a value \( h = g^x \). The leakage offered by this oracle (and indeed the existence of this leakage) is not addressed in A²L’s proof of security.

Key Recovery Attack. In our first attack, we completely recover the decryption key \( \mathsf{dk}_H \) of the hub by simply querying the oracle \( \Pi_{O^{\mathcal{L}}, \mathsf{ds}, \mathsf{HL}, \mathsf{I}, \mathsf{ADP}} \) \( n \) times. For this attack, we assume that the encryption scheme \( \Pi_E \) is (in addition to being re-randomizable and CPA-secure as required by A²L):

- Linearly homomorphic over \( \mathcal{G} \).
- Circular secure for bit encryption, i.e., the scheme is CPA-secure even given the bitwise encryption of the decryption key \( \text{Enc}(\mathsf{ek}_H, \mathsf{dk}_1), \ldots, \text{Enc}(\mathsf{ek}_H, \mathsf{dk}_n) \).
- The above-mentioned ciphertexts \((c_1, \ldots, c_n) := (\text{Enc}(\mathsf{ek}_H, \mathsf{dk}_1), \ldots, \text{Enc}(\mathsf{ek}_H, \mathsf{dk}_n)) \) are included in the encryption key \( \mathsf{ek}_H \).

Such schemes can be constructed from a variety of standard assumptions [13]. It is easy to see that these additional requirements do not contradict the initial prerequisites of the scheme.

The attack is shown in Algorithm 1. Note that, for a signing key pair in the \( i \)-th iteration, if the \( O^{\mathcal{L}} \) oracle returns \( y \neq \bot \), this means that in the coin mixing layer, the Hub has obtained a valid \( y \) and thus
Algorithm 1 Key Recovery Attack

Input: Hub’s ek along with the ciphertexts \( (c_1, \ldots, c_n) \)

1. Initialize key guess \( dk' := 0^n \)
2. for \( i = 1 \ldots n \) do
3. Sample \( x \sim \mathbb{Z}_p \) and compute \( h := g^x \)
4. Sample a fresh signing key \( (vk, sk) \leftarrow \text{KGen}(t^p) \)
5. Set \( c_i' := \Pi_E.\text{Enc}(ek, x) \circ c_i = \Pi_E.\text{Enc}(ek, x + dk_i) \)
6. Compute \( \alpha_i \leftarrow \Pi_{\text{ADP}}.\text{PreSig}(sk, m, h) \)
7. Query \( y \leftarrow \text{O}^{\Pi_L}_{\text{ek}}(vk, m, h, c_i', \alpha_i) \)
8. If \( y = \bot \) set \( dk'_i := 1 \)
end for
return \( dk' \)

Algorithm 2 One-More Signature Attack

Input: Bob’s ciphertexts \( (c_1, \ldots, c_{q+1}) \) and group elements \( (h_1, \ldots, h_{q+1}) \), where \( c_j = \oplus_{i=1}^n \Pi_E.\text{Enc}(ek, x_j) \) and \( h_j := g^{x_j} \) and Hub’s ek

1. Initialize guess \( x'_1 = 0^n \) and a counter \( i = 1 \)
2. for \( i = 1 \ldots n \) do
3. Sample a fresh signing key \( (vk, sk) \leftarrow \text{KGen}(t^p) \)
4. Compute \( c'_i := \Pi_E.\text{CFlip}(ek, l, c_i) \)
5. Sample \( (r_1, \ldots, r_{q+1}) \leftarrow \mathbb{Z}_p^q \)
6. Compute \( c' := (c'_1)^{r_1} \circ (c'_2)^{r_2} \circ \cdots \circ (c'_{q+1})^{r_{q+1}} \)
7. Compute \( h' := \prod_{j=1}^{q+1} h_j^{r_j} \)
8. Sign \( \sigma' \leftarrow \Pi_{\text{ADP}}.\text{PreSig}(sk, m, h') \)
9. Query \( y_i \leftarrow \text{O}^{\Pi_L}_{\text{ek}}(vk, m', h', c', \sigma') \)
10. If \( y_i = \bot \) set \( x'_1[i] = 1 \)
end for
11. Continue querying (without updating \( x'_1 \)) until \( q \) non-\( \bot \) queries have been made
12. For all \( i \) corresponding to a non-\( \bot \) query, set \( E_i \) to be the equation \( y_i - x'_1[i] = r_1 x'_1[1] + \cdots + r_{q+1} x'_{q+1} \)
13. Solve \( (E_1, \ldots, E_q) \) for \( (x'_2, \ldots, x'_{q+1}) \)
14. return \( (x'_1, x'_2, \ldots, x'_{q+1}) \)

Obtains Alice’s (adversary’s) signature on a transaction. Due to one-time use of keys in this (cryptocurrency) layer, the attacker therefore cannot reuse the same signing key pair in another iteration for a different message (transaction). Therefore, it is necessary that the attacker (Alice) sample different keys to account for every iteration being a non-\( \bot \) query to \( \text{O}^{\Pi_L} \). This is realized in the real world by the attacker having \( n \) different sessions (of coin mixing), one for each \( v_k \), with Hub.

Observe that the response of the oracle is \( \bot \) if and only if \( dk_1 = 1 \), since \( h = g^x \neq g^{x+1} \). On the other hand, if \( dk_1 = 0 \), then the oracle always returns a valid adapted signature \( \sigma' \). Thus, the attack succeeds with probability 1.

One-More Signature Attack. We present a different attack, where we impose different assumptions on the encryption scheme \( \Pi_E \). We discuss later in the section why these assumptions do not contradict the pre-requisites of the \( A^L \) scheme. Specifically, in addition to \( A^L \)’s requirement that the scheme is perfectly re-randomizable and CPA-secure, we assume that it is:
- Linearly homomorphic over \( \mathbb{Z}_p \).
- Supports homomorphic evaluation of the conditional bit flip (CFlip) function, defined as

\[
\Pi_E.\text{CFlip}(ek, i, \text{Enc}(ek, x)) := \text{Enc}(ek, y) \text{ where } \begin{cases} y = x & \text{if } x_i = 0 \\ y = x \oplus e_i & \text{if } x_i = 1 \end{cases}
\]

and \( e_i \) is the \( i \)-th unit vector.

The objective of the attack is to steal coins from the hub in the coin mixing protocol. Specifically, at the \( A^L \) level, the attacker will solve \( q + 1 \) puzzles by querying the puzzle solver interface successfully only \( q \) times. Note that we do not count unsuccessful (i.e., the oracle returns \( \bot \)) queries, since those non-accepting queries do not correspond to any payment from Alice’s side.

The attack is shown in Algorithm 2. We assume (for convenience) that \( q \geq n \) and that \( \mathbb{Z}_p \leq 2^n \) and therefore \( x_j \in \{0, 1\}^n \). Observe that the attack makes at most \( q \) successful queries to the oracle, so all we need to show is that the success probability is high enough. First, we argue that the attack recovers the correct \( x'_1 \) with probability 1. If the \( i \)-th bit \( x_{1,i} = 0 \), then the CFlip operation does not alter the content of the ciphertext and therefore

\[
c'_i = \text{Enc} \left( ek, \sum_{j=1}^{q+1} r_j(i) \cdot x_j \right) \text{ and } h' = \prod_{j=1}^{q+1} h_j^{r_j(i)} = g^{x'_1(i)} \cdot x_j
\]

so the oracle always returns a non-\( \bot \) response. On the other hand, if \( x_{1,i} = 1 \), then the above equality does not hold and therefore \( \text{O}^{\Pi_L}_{\text{ek}}.\text{ADP} \) always returns \( \bot \).

This querying strategy is repeated for every bit of \( x'_1 \) and continued on \( x_2 \), etc., until \( q \) non-\( \bot \) queries have been made. Because \( q \geq n \), the attacker will have learned all \( n \) bits of \( x'_1 \) by this point. Thus, the set of equations \( (E_1, \ldots, E_q) \) has exactly \( q \) unknowns. Since the coefficients are uniformly chosen, the equations are, with all but negligible probability, linearly independent. Since \( \mathbb{Z}_p \) is a field, the solution is uniquely determined and can be found efficiently via Gaussian elimination.

N-More Signatures. The described attack is in fact even stronger than shown. Using this method, an attacker \( A \) can use \( q \) queries, where \( [q] = \mathbb{N}_q \), to recover \( N + q \) plaintexts. \( A \) does this by using \( N \) queries to recover the first \( N \) plaintexts \( x_1, \ldots, x_N \) and \( N \) equations as described previously (once it has flipped all \( n \) bits in \( x_1 \), it starts flipping bits in \( x_2 \), and so on). Using its remaining queries, it obtains \( q - N \) more equations (either by continuing to flip bits in further ciphertexts, which are however wasted, or by simply choosing new values \( r_i \) for the linear combinations) for a total of \( q \) equations. Using Gaussian elimination, it can recover the remaining \( q \) plaintexts \( x_{N+1}, \ldots, x_{N+q} \). Taken with the plaintexts \( x_1, \ldots, x_N \) that were recovered bit-by-bit, the attacker has learned \( N + q \) plaintexts.

Instantiations. We now justify our additional assumptions on the encryption scheme \( \Pi_E \) by describing suitable instantiations that satisfy all the requirements. Clearly, if the scheme is fully-homomorphic [25] then it supports both linear functions over \( \mathbb{Z}_p \) and conditional bit flips. However, we show that even a linear homomorphic encryption (over \( \mathbb{Z}_p \)) can suffice to mount our attack. Specifically, given a CPA-secure linearly homomorphic encryption scheme (KGen\( ^* \), Enc\( ^* \), Dec\( ^* \)), we define a bitwise encryption scheme (KGen, Enc, Dec) as follows:
A blind conditional signature (BCS) is executed among users Alice, Bob, and Hub. The interfaces and associated security properties are defined below.

Definition 4.1 (Blind Conditional Signature). A blind conditional signature $\Pi_{\text{BCS}} := (\text{Setup}, \text{PPromise}, \text{PSolver}, \text{Open})$ is defined with respect to a signature scheme $\Pi_{\text{DS}} := (\text{KGen}, \text{Sign}, \text{Vf})$ and consists of the following efficient algorithms.

- $(\text{ek, dk}) \leftarrow \text{Setup}(1^n)$: The setup algorithm takes as input the security parameter $1^n$ and outputs a key pair $(\text{ek}, \text{dk})$.
- $(\bot, \{\tau, \bot\}) \leftarrow \text{PPromise}(H(\text{dk}, \text{sk}_H^{\tau}, m_{\text{HB}}))$: The puzzle promise algorithm is an interactive protocol between two users $H$ (with inputs the decryption key $\text{dk}$, the signing key $\text{sk}_H$, and a message $m_{\text{HB}}$) and $B$ (with inputs the encryption key $\text{ek}$, the verification key $\text{vk}_H$, and a message $m_{\text{HB}}$) and returns $\bot$ to $H$ and either a puzzle $\tau$ or $\bot$ to $B$.

Next, we define correctness.

Definition 4.2 (Correctness). A blind conditional signature $\Pi_{\text{BCS}}$ is correct if for all $n \in \mathbb{N}$, all $(\text{ek}, \text{dk})$ in the support of $\text{Setup}(1^n)$, all $(\text{vk}_H^A, \text{sk}_A^H)$ and $(\text{vk}_A^B, \text{sk}_A^B)$ in the support of $\Pi_{\text{DS}}. \text{KGen}(1^n)$, and all pairs of messages $(m_{\text{HB}}, m_{\text{AH}})$, it holds that

$$\Pr[Vf(\text{vk}_H^A, m_{\text{HB}}, \text{Open}(\tau, s)) = 1] = 1$$

and

$$\Pr[Vf(\text{vk}_A^B, m_{\text{AH}}, \sigma^*) = 1] = 1$$

where

- $\tau \leftarrow \text{PPromise}(H(\text{dk}, \text{sk}_H^{\tau}, m_{\text{HB}}))$ and
- $((\sigma^*, s), \sigma^*) \leftarrow \text{PSolver}(A(\text{sk}_A^A, \text{ek}, m_{\text{AH}}, \tau))$.

We now present the security guarantees of BCS in the game-based setting. Our definition of blindness is akin to the strong blindness notion of standard blind signatures [19], in which the adversary picks the keys (as opposed to the weak version in which they are chosen by the experiment)\(^6\). Roughly speaking, it says that two promise/solve sessions cannot be linked together by the hub.\(^6\)

Definition 4.3 (Blindness). A blind conditional signature $\Pi_{\text{BCS}}$ is blind if there exists a negligible function $\negl(n)$ such that for all $n \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, the following holds:

$$\Pr[\text{ExpBlnd}_{\Pi_{\text{BCS}}}(\mathcal{A}) = 1] \leq \frac{1}{2} + \negl(n)$$

where $\text{ExpBlnd}$ is defined in Figure 5.\(^7\)

Next, we define unlockability, which says that it should be hard for Hub to create a valid signature on Alice’s message that does not allow Bob to unlock the full signature in the corresponding promise session.

\(^3\)We opt for this stronger version since we want to provide anonymity even in the case of a fully malicious hub, which can pick its keys adversarially to try to link a sender/receiver pair.

\(^6\)We do not consider the case in which Hub colludes with either Alice or Bob, since deanonymization is trivial (Alice (resp. Bob) simply reveals the identity of Bob (resp. Alice) to Hub); this is in line with [33].

\(^7\)In previous works, descriptions of unlinkability assume an explicit step for blinding the puzzle $\tau$ between PPOM and PSolver. Here, we assume that PSolver performs this blinding functionality.
A blind conditional signature $\Pi_{BCS}$ is unlockable if there exists a negligible function $\text{negl}(n)$ such that for all $n \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, the following holds:

$$\Pr\left[\text{ExpUnlock}^{\mathcal{A}}_{\Pi_{BCS}}(n) = 1\right] \leq \text{negl}(n)$$

where ExpUnlock is defined in Figure 6.

Our definition of unforgeability is inspired by the unforgeability of blind signatures [19]. We require that Alice and Bob cannot recover $q$ signatures from Hub while successfully querying the solving oracle at most $q - 1$ times. Since each successful query reveals a signature from Alice’s key (which in turn corresponds to a transaction from Alice to Hub), this requirement implicitly captures the fact that Alice and Bob cannot steal coins from Hub. The winning condition $b_0$ captures the scenario where the adversary forgives a signature of the hub on a message previously not used in any promise oracle query. The remaining conditions $b_1, b_2$ and $b_3$ together capture the scenario in which the adversary outputs $q$ valid distinct key-message-signature tuples while having queried for solve only $q - 1$ times. Hence, in the second condition, the attacker manages to complete $q$ promise interactions with only $q - 1$ solve interactions, whereas in the first winning condition, the adversary computes a fresh signature that is not the completion of any promise interaction. These conditions are technically incomparable: an attacker that succeeds under one condition does not imply an attacker succeeding on the other. It is important to note that this is different from the unforgeability notion of blind signatures (where the attacker only has access to a single signing oracle), since in our case the hub is offering the attacker two oracles: promise and solve.

**Definition 4.5 (Unforgeability).** A blind conditional signature $\Pi_{BCS}$ is **unforgeable** if there exists a negligible function $\text{negl}(n)$ such that for all $n \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, the following holds:

$$\Pr\left[\text{ExpUnforg}^{\mathcal{A}}_{\Pi_{BCS}}(n) = 1\right] \leq \text{negl}(n)$$

where ExpUnforg is defined in Figure 7.

We define security as the collection of all properties.

**Definition 4.6 (Security).** A blind conditional signature $\Pi_{BCS}$ is **secure** if it is blind, unlockable, and unforgeable.

### 4.2 The A²L⁺ Protocol

In the following we describe our A²L⁺ construction. Our scheme is a provable variant of A²L (Section 3.1) and therefore we only describe the differences with respect to the original protocol. The concrete modifications are as follows:

- Augment the public key of Hub $ek\mathcal{H}$ with a NIZK proof that certifies that $ek\mathcal{H} \in \text{Supp}(\Pi_{BCS}.\text{KGen}(1^n))$. All parties verify this proof during their first interaction with Hub.
- In PSolver (Figure 13), Hub additionally checks if $vk_{\mathcal{H}}^{\mathcal{A}}$ is in the support of $\Pi_{\text{ADP}}.\text{KGen}(1^n)$ before the decryption (line 6). Furthermore, we replace the condition (line 8) with

$$\Pi_{\text{ADP}}.\text{PreVf}(vk_{\mathcal{H}}^{\mathcal{A}}, m_{\mathcal{A}H}, Y', \hat{\sigma}_{\mathcal{A}H}) \neq 1 \lor g^w \neq Y'.$$

### 4.3 Security Analysis

In this section we present our security results and defer the proofs to Appendix C. Before proving our main theorem, we define a property which is going to be useful for our analysis.

**Definition 4.7 (OM-CCA-A2L).** An encryption scheme $\Pi_E$ is one-more CCA-A2L-secure (OM-CCA-A2L) if there exists a negligible function $\text{negl}(n)$ such that for all $n \in \mathbb{N}$, all polynomials $q = q(n)$, and all PPT adversaries $\mathcal{A}$, the following holds:

$$\Pr\left[\text{OM-CCA-A2L}^{\mathcal{A}}_{\Pi_E.q}(n) = 1\right] \leq \text{negl}(n),$$
The following technical lemma shows that an LOE scheme satisfies this property, assuming the hardness of the OMDL problem. The formal analysis of the below lemma is deferred to Appendix C.

**Lemma 4.8.** Let \( \Pi_E \) be an LOE scheme. Assuming the hardness of OMDL, \( \Pi_E \) is OM-CCA-A2L secure.

**Main Theorem.** We are now ready to give the main theorem of this section. The formal analysis is deferred to Appendix C.

**Theorem 4.9.** Let \( \Pi_E \) be an LOE scheme, \( \Pi_{\text{ADP}} \) a secure adaptor signature scheme, and \( \Pi_{\text{NIZK}} \) a sound NIZK proof system. Assuming the hardness of OMDL, the \( \mathcal{A}_L^2L^* \) protocol is a secure blind conditional signature scheme.

## 5 UC-SECURE BLIND CONDITIONAL SIGNATURES

We now model security in the universal composability framework from Canetti [16] extended to support a global setup [17] in order to capture concurrent executions. We refer the reader to [16] for a comprehensive discussion. We consider static corruptions, where the adversary announces at the beginning which parties it corrupts. We denote the environment by \( E \). For a real protocol \( \Pi \) and an adversary \( A \) we write \( \text{EXEC}_{\Pi, A, E} \) to denote the ensemble corresponding to the protocol execution. For an ideal functionality \( \mathcal{F} \) and an adversary \( S \) we write \( \text{EXEC}_{\mathcal{F}, S, E} \) to denote the distribution ensemble of the ideal world execution.

**Definition 5.1 (Universal Composability).** A protocol \( \Pi \) UC-realizes an ideal functionality \( \mathcal{F} \) if for any PPT adversary \( A \) there exists a simulator \( S \) such that for any environment \( E \) the ensembles \( \text{EXEC}_{\Pi, A, E} \approx \text{EXEC}_{\mathcal{F}, S, E} \) are computationally indistinguishable.

In our protocol, we assume the existence of a general-purpose UC-secure 2-party computation (2PC) protocol [18, 33], where two parties interact with the ideal functionality to compute a function \( f(x, y) \) over their private inputs \( x \) and \( y \).

### 5.1 Ideal functionality

We describe the ideal functionality \( \mathcal{F}_{\text{BCS}} \) that captures the functionality and security of BCS in the UC framework. We refer the reader to Appendix D for the formal description of \( \mathcal{F}_{\text{BCS}} \). The ideal functionality has three routines, namely for puzzle promise, puzzle solve, and open, which intuitively capture the functionality of BCS as discussed in Section 4. On a high level, \( \mathcal{F}_{\text{BCS}} \) captures blindness by sampling the puzzle identifiers pid and pid′, which correspond to puzzle promise and puzzle solve interactions, locally together, but never revealing them together to the hub. \( \mathcal{F}_{\text{BCS}} \) captures atomicity by returning a successful message (not aborting) for pid during open if and only if it sent a successful solved message during the puzzle solve interaction for the puzzle identifier pid′ (where pid and pid′ correspond to each other). Note that the above atomicity guarantee implies the game-based definitions of unlockability and unforgeability.

Our functionality \( \mathcal{F}_{\text{BCS}} \) is taken verbatim from the \( \mathcal{F}_{\mathcal{A}_L^2L} \) functionality in [53] except that we do not consider user registrations (as done in \( \mathcal{F}_{\mathcal{A}_L^2L} \)) to tackle grieving attacks [44] in the coin mixing layer. These attacks are mounted by Bob starting many puzzle promise operations, each of which requires Hub to lock coins, whereas the
corresponding puzzle solver interactions are never carried out. As a consequence, all of Hub’s coins are locked and no longer available, which results in a form of denial of service attack. We argue that the issue does not concern the functionality or security of BCS as a cryptographic tool, but only affects the coin mixing protocol at the transaction layer. We emphasize that griefing attacks can be thwarted at this layer in both the formal model and the construction using the same ideas as in [53].

### 5.2 Our Protocol: A₂²\text{UC}

We now describe our protocol A₂²\text{UC} that realizes the ideal functionality \(f_{\text{BCS}}\). We assume the following cryptographic building blocks:

- An adaptor signature scheme \(\Pi_{\text{ADP}}\) defined with respect to \(\Pi_{\text{DS}}\) and a hard relation \(R_{\text{DL}}\).
- A UC-secure NIZK proof system \(\Pi_{\text{NIZK}}\) for the language
  \(L := \{(ek, Y, c) \mid \exists s, \text{s.t. } c \leftarrow \Pi_{\text{E}}.\text{Enc}(ek, s) \land Y = g^s\}\).
- A UC-secure 2PC protocol.
- A CCA-secure [8] encryption scheme \(\Pi_{\text{E}} := (\text{KGen}, \text{Enc}, \text{Dec})\) with unique decryption keys.

The property of unique decryption keys is formalized below.

**Definition 5.2 (Unique Decryption Keys).** An encryption scheme \(\Pi_{\text{E}}\) has unique decryption keys if the KGen algorithm is of the following form:

- Sample \(dk \leftarrow \{0, 1\}^n\).
- Run \(ek \leftarrow \text{Gen}(dk)\).

Furthermore, for all \(ek\) output by \(\text{KGen}\), there exists a unique \(dk\) such that \(ek = \text{Gen}(dk)\). In other words, \(\text{Gen}\) is injective.

This property is already satisfied by most natural public-key encryption schemes, but it can be generically achieved by augmenting the encryption key with a perfectly binding commitment \(\text{com}(dk)\) to the decryption key \(dk\).

**Protocol Description.** We assume Alice and Hub have a key pair for the signature scheme \(\Pi_{\text{DS}}\). Specifically, we have the verification-signing key pair \((vk_{\text{DB}}, sk_{\text{DB}})\) and \((vk_{\text{AH}}, sk_{\text{AH}})\), belonging to Hub and Alice, respectively. We then have two messages \(m := m_{\text{DB}}\) and \(m' := m_{\text{AH}}\) for which the users wish to generate blind conditional signatures. The setup and open algorithms are formally described in Figure 9. The puzzle promise and puzzle solver of A₂²\text{UC} are formally described in Figure 10 and Figure 11, respectively. For ease of understanding, we briefly describe below our A₂²\text{UC} protocol in terms of the differences with the A²\text{L} protocol (Figures 12 and 13).

- The setup algorithm (Figure 9) of A₂²\text{UC} generates the keys of Hub, which are the keys for the (CCA-secure) encryption scheme \(\Pi_{\text{E}}\).

- In PPromise of A₂²\text{UC} (Figure 10),
  - The NIZK proof system is UC-secure.
  - Bob no longer re-randomizes the instance or the ciphertext. Therefore, we drop the re-randomization steps (line 9 and 10) of PPromise in A²\text{L} (Figure 12). Simply set the puzzle to \(\tau := (m_{\text{DB}}, \tilde{\sigma}_{\text{AH}}(Y, c))\).

- In PSolver of A₂²\text{UC} (Figure 11),
  - Alice no longer sends the ciphertext to Hub (line 5 of Figure 13). We therefore remove the local decryption step (line 6 of Figure 13), and replace it with a 2PC protocol (line 6 of Figure 11).
  - At the end of the 2PC protocol, Alice receives \(\perp\), while Hub receives the value \(z\). Hub additionally checks if \(Y' = g^z\) (line 7) and uses \(z\) to adapt the pre-signature \(\tilde{\sigma}_{\text{AH}}\) to signature \(\sigma_{\text{AH}}\).
  - We add a check for Alice (line 10) that \(\sigma_{\text{AH}}\) is a valid signature before extracting the witness \(z'\) in line 12.

- The Open algorithm (Figure 9) is the same as in Figure 14 of A²\text{L}, except we skip removing the randomness factor. The algorithm in Figure 9 now simply adapts a pre-signature \(\tilde{\sigma}\) to a valid signature \(\sigma\) which it returns as output.

### 5.3 Security Analysis

We now show that A₂²\text{UC} satisfies UC-security. In favor of a simpler analysis, we assume that the verification keys of all parties are honestly generated. In practice, this can be enforced by augmenting keys with NIZKs that certify their validity [12, 36]. We state here our security theorem and defer the formal proof to Appendix C.

**Theorem 5.3.** Let \(\Pi_{\text{E}}\) be a CCA-secure encryption scheme, \(\Pi_{\text{ADP}}\) a secure adaptor signature scheme, 2PC a UC-secure two-party computation protocol, and \(\Pi_{\text{NIZK}}\) a UC-secure NIZK for the language \(L\) above. Then the A₂²\text{UC} protocol UC-realizes \(f_{\text{BCS}}\).

### 6 Efficiency

We now discuss the efficiency of our constructions A²²\text{L}⁺ and A₂²\text{UC} in terms of number of cryptographic operations.

#### 6.1 A²²\text{L}⁺

Recall that we use an encryption scheme \(\Pi_{\text{E}}\) in the LOE model. Below we present an instantiation of such a \(\Pi_{\text{E}}\).

**Instantiating Linear-Only Encryption.** As shown in [11] it is not sufficient to instantiate this with any linearly homomorphic encryption (e.g., ElGamal). Though the scheme may not support homomorphic operations beyond linear, it may still have obliviously sampleable ciphertexts, i.e., the ability to sample a ciphertext without knowing the underlying plaintext. Note that this falls outside the LOE model, since there is no oracle that implements this functionality. Thus, as suggested in [11] we implement an additional safeguard needed to prevent oblivious sampling. Given a linearly homomorphic encryption scheme \(\Pi_{\text{E}}^\perp := (\text{KGen}^\perp, \text{Enc}^\perp, \text{Dec}^\perp)\) over \(\mathbb{Z}_p\), we define a candidate LOE \(\Pi_{\text{E}} := (\text{KGen}, \text{Enc}, \text{Dec})\) as follows:

```
<table>
<thead>
<tr>
<th>Setup((1^n))</th>
<th>Open((\tau, s))</th>
</tr>
</thead>
<tbody>
<tr>
<td>({ek, dk} \leftarrow \Pi_{\text{E}}.\text{KGen}((1^n)))</td>
<td>(\text{parse } \tau := (\gamma, \sigma, \cdot))</td>
</tr>
<tr>
<td>set (pk := ek, sk := dk)</td>
<td>(\sigma \leftarrow \Pi_{\text{ADP}}.\text{Adapt}(\gamma, s))</td>
</tr>
<tr>
<td>return ((pk, sk))</td>
<td>return (\sigma)</td>
</tr>
</tbody>
</table>
```
Figure 10: Puzzle promise protocol of $A^2L^{UC}$

Table 1: Operations in $A^L$ and $A^L^*$ when instantiated with Schnorr or ECDSA adaptor signatures [4]. We give the number group exponents (Exp) and group operations (Op) in both class groups (CL) and groups of prime order $p$ ($\mathbb{G}$), where $\log p = n$. Group element inversions (Inv) only occur in class groups. Modular multiplications ($\times$) and additions ($+$) are performed modulo $q$. We denote by #H the number of hash computations. Decryption of a CL ciphertext also involves solving a discrete logarithm in a class group, which we denote by DLog.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Signature</th>
<th>Exp (CL)</th>
<th>Op (CL)</th>
<th>Inv (CL)</th>
<th>DLog (CL)</th>
<th>Exp ($\mathbb{G}$)</th>
<th>Op ($\mathbb{G}$)</th>
<th>$\times$ mod $q$</th>
<th>$+$ mod $q$</th>
<th>#H</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A^2L$ (insecure)</td>
<td>Schnorr</td>
<td>18</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>13</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>ECDSA</td>
<td>18</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>27</td>
<td>8</td>
<td>17</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$A^2L^*$</td>
<td>Schnorr</td>
<td>28</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>14</td>
<td>9</td>
<td>5</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>ECDSA</td>
<td>28</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>32</td>
<td>10</td>
<td>21</td>
<td>12</td>
<td>11</td>
</tr>
</tbody>
</table>
We investigate the notion of synchronization puzzles, the cryptographic building blocks at the core of hub-enabled coin mixing services. We find that the previous formalization of a synchronization puzzle in [53] is flawed. In fact, we identify several issues in its formal model which can be easily exploited to break the security of the resultant coin mixing protocol. We conclude that tighter formalization of the functionality and security of synchronization puzzles is necessary.

To fill this gap, we introduce the notion of blind conditional signatures (BCS). Additionally, we provide different security formalizations for BCS at varying levels of strength (game-based and in the UC framework) accompanied by a provably secure variant of A^2L called A^2L^+ and a new provably UC-secure construction A^2L^UC. Our performance evaluation results show an efficiency vs. security trade-off in the case of our constructions, yet show with A^2L^+ that provably secure coin mixing services are deployable in practice.

ACKNOWLEDGMENTS

This work was partially supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research grant agreement 771527-BROWSEC, by the Austrian Science Fund (FWF) through the projects PROFET (grant agreement P31621) and the project W1255-N23, by the Austrian Research Promotion Agency (FGF) through COMET K1 SBA and COMET K1 ABC, by the Vienna Business Agency through the project Vienna Cybersecurity and Privacy Research Center (VISI), by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association through the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT). This work has also been partially supported by the Madrid regional government as part of the program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union, by grant IJC2020-043391-I/UCIN/AEU/10.13039/501100011033 and European Union NextGenerationEU/PRTR, by SCUM Project (RTI2018-102043-B-I00) MCIN/AEU/10.13039/501100011033/ERDF A way of making Europe, and by the project HACrypt. Additionally, this material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 1840340. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The work was also made possible through the support of the David and Lucille Packard Foundation (award #20207130), SRI International (award #53978) / Prime: Defense Advanced Research Projects Agency (award #HR00110C0086), and the National Science Foundation (award #2212746). G.M. was partially supported by the German Federal Ministry of Education and Research (BMBF) (grant 16K15K042, project 6GEM).

REFERENCES

A RANDOMIZABLE PUZZLES AND HOMOMORPHIC ENCRYPTION

Here we recall the definitions of randomizable puzzles [53] and we show that they are trivially satisfied by a CPA-secure homomorphic encryption (over $\mathbb{Z}_p$), with statistical circuit privacy [42]. We recall the syntax as defined in [53].

**Definition A.1 (Randomizable Puzzle).** A randomizable puzzle scheme $P = \{\text{PSetup}, \text{PGen}, \text{PSolve}, \text{PRand}\}$ with a solution space $S$ (and a function $\phi$ acting on $S$) consists of four algorithms defined as:

1. $(pp, td) \leftarrow \text{PSetup}(1^n)$: a PPT algorithm that on input security parameter $1^n$, outputs public parameters $pp$ and a trapdoor $td$.
2. $Z \leftarrow \text{PGen}(pp, \xi)$: a PPT algorithm that on input public parameters $pp$ and a puzzle solution $\xi$, outputs a puzzle $Z$.
3. $\xi := \text{PSolve}(td, Z)$: a PPT algorithm that on input a trapdoor $td$ and puzzle $Z$, outputs a puzzle solution $\xi$.
4. $(Z', r) \leftarrow \text{PRand}(pp, Z)$: a PPT algorithm that on input public parameters $pp$ and a puzzle $Z$ (which has a solution $\xi$), outputs a randomization factor $r$ and a randomized puzzle $Z'$ (which has a solution $\phi(\xi, r)$).

It is not hard to see that a linearly homomorphic encryption scheme $(\text{KGen}, \text{Enc}, \text{Dec})$ matches the syntax of a randomizable puzzle, setting $pp$ to the encryption key and $td$ to be the decryption key. For the $\text{PRand}$ algorithm, we can sample a random $r \xleftarrow{\$} \mathbb{Z}_p$ and compute

$$\text{Enc}(ek, \xi) \circ \text{Enc}(ek, r) = c$$

which is an encryption of $\phi(\xi, r) = \xi + r$. Next we recall the definition of security for randomizable puzzles.

**Definition A.2 (Security).** A randomizable puzzle scheme $P$ is secure, if there exists a negligible function $\text{negl}(\cdot)$ such that

$$\Pr \left[ \xi \leftarrow \mathcal{A}(pp, Z) \mid (pp, td) \leftarrow \text{PSetup}(1^n), \xi \leftarrow S, Z \leftarrow \text{PGen}(pp, \xi) \right] \leq \text{negl}(n).$$

This follows as an immediate application of CPA-security (in fact, even the weaker one-wayness suffices) of the encryption scheme. Finally we recall the notion of privacy.

**Definition A.3 (Privacy).** A randomizable puzzle scheme $P$ is private if for every PPT adversary $\mathcal{A}$ there exists a negligible function $\text{negl}(\cdot)$ such that

$$\Pr[\mathcal{A}^{\text{RandSec}_{\mathcal{A},P}}(n) = 1] \leq 1/2 + \text{negl}(n)$$

where the experiment $\mathcal{A}^{\text{RandSec}_{\mathcal{A},P}}$ is defined as follows:

- $(pp, td) \leftarrow \text{PSetup}(1^n)$
- $((Z_0, q_0), (Z_1, q_1)) \leftarrow \mathcal{A}(pp, td)$
- $b \xleftarrow{\$} \{0, 1\}$
- $(Z_b^q, r_0) \leftarrow \text{PRand}(pp, Z_0)$
- $(Z_1^q, r_1) \leftarrow \text{PRand}(pp, Z_1)$
- $b' \leftarrow \mathcal{A}(pp, td, Z_b^q)$
- Return $\text{PSolve}(td, Z_b) = \xi_0 \wedge \text{PSolve}(td, Z_1) = \xi_1$ and $b \equiv b'$

Recall that circuit privacy implies that the distribution induced by $\text{Enc}(ek, \xi) \circ \text{Enc}(ek, r)$ is statistically close to that induced by a fresh encryption $\text{Enc}(ek, \xi + r)$. This implies that privacy is satisfied in a statistical sense. Thus we can state the following.

**Lemma A.4.** Assuming that $(\text{KGen}, \text{Enc}, \text{Dec})$ is a linearly homomorphic encryption with statistical circuit privacy, there exists a randomizable puzzle with statistical privacy.

B A²L PROTOCOL

We recall the formal description of Puzzle promise (Figure 12) and Puzzle solver (Figure 13) of $A^2L$. We report their protocol after translating the same to our syntax for consistency. The setup algorithm is the same as in Figure 9 and the open algorithm is given in Figure 14.

C SECURITY PROOFS

**Proof of Lemma 4.8.**

**Proof.** We give a proof by reduction. Let $\mathcal{A}$ be a PPT adversary with non-negligible advantage in the OMDL-CCA game. We now construct an adversary $\mathcal{R}$ which uses $\mathcal{A}$ to break the security of OMDL.

$\mathcal{R}$ is given $(h_1, \ldots, h_{q+1}) = (g^r, \ldots, g^{q+1})$ by the OMDL game. It will run $\mathcal{A}$ to attempt to obtain the $q + 1$ discrete logarithms to win the game. Crucially, $\mathcal{R}$ must simulate $\mathcal{A}$’s oracle access to $O^{\mathcal{A},L}_{\text{KGen,Enc,Dec}}$, which consists of at most $q$ successful queries (but unlimited $\bot$ queries), while making at most $q$ queries (of any kind) to its oracle $DL(\cdot)$.

$\mathcal{R}$ proceeds as follows. First, it samples $q + 1$ uniform $\lambda$-bit strings $(c_1^1, \ldots, c_{q+1}^1)$. Note that these are identically distributed to outputs of $O^{\text{Enc}}$. It enters $(X_1, c_1^1), \ldots, (X_{q+1}, c_{q+1}^1)$ into a table $M$, where the $X_i$ are random variables. Now it sends $(c_1^1, h_1), \ldots, (c_{q+1}^1, h_{q+1})$ to the adversary $\mathcal{A}$.

Any queries $\mathcal{A}$ makes to the encryption scheme oracles $(O^{\text{Gen}}, O^{\text{Enc}}, O^{\text{Dec}}, O^{\text{Adv}})$ and their corresponding responses are passed along unchanged by $\mathcal{R}$ but recorded in its table $M$. Whenever $\mathcal{A}$ makes some query $(vk_i, m_i, k_i, \sigma_i, \tilde{\sigma}_i)$ to $O^{\text{Adv}}$, $\mathcal{R}$ first checks that $vk_i$ is in the support of $\Pi_{\text{Adv},\text{KGen}}(1^n)$ (this is a publicly checkable predicate since the valid verification keys are defined to be all group elements). After this, it acts in one of four ways:

1. If $c_i = c_i^1$ and $k_i = h_j$ for some $j$, it checks $\text{PreVf}(vk_i, m_i, k_i, c_i^1) = 1$. If not, it returns $\bot$; otherwise, it queries $DL(h_j)$ to get $x_j$ and returns $\Pi_{\text{Adv},\text{Adapt}}(\sigma_i, x_j)$ to $\mathcal{A}$.
2. If $c_i = c_i^1$ but $k_i \neq h_j$, $\mathcal{R}$ sends $\bot$ to $\mathcal{A}$.
3. If $(\cdot, c_i) \notin M$, $\mathcal{R}$ sends $\bot$ to $\mathcal{A}$.
Otherwise, let $p(a)$ if

Notice that, by the linear-only property of the encryption

deg the output to

if the pre-signature verifies and adapts it if so) and sends

$x$

Public parameters: group description $(G, g, q)$, message $m_{IB}$

\[ \Pi_{ADP}.PreSig(\hat{sk}^H_{IB}, m_{IB}, Y) \]

\[ Y, c, \pi, \hat{\sigma}^H \]

If $\Pi_{ADP}.PreVf(\hat{sk}^H_{IB}, m_{IB}, Y, \hat{\sigma}^H) \neq 1$ then return \perp

If $\Pi_{ADP}.PreVf(\hat{sk}^H_{IB}, m_{IB}, Y, \hat{\sigma}^H) = 1$ then return \perp

\[ r \rightarrow s \pi, Y' = Y \cdot g^r \]

$\epsilon' \leftarrow \Pi_{E}.Rand(\epsilon, r)$

Set $r = (r, m_{IB}, \hat{\sigma}^H (Y, c), (Y', \epsilon'))$

return $r$

---

**Figure 12: Puzzle promise protocol of $A^2L$**

---

**Figure 13: Puzzle solver protocol of $A^2L$**

---

**Figure 14: Open algorithm of $A^2L$**

(4) Otherwise, let $p_i$ be the plaintext entry corresponding to $c_i$ in $M$. Notice that, by the linear-only property of the encryption scheme, $p_i$ is a polynomial in $X_1, \ldots, X_{q+1}$ with $\deg(p_i) \leq 1$.

(a) If $\deg(p_i) = 0$, $p_i$ is some constant value $x_j$. In this case, $R$ uses $x_j$ to proceed as the normal $A^{2L}$ oracle does (checks if the pre-signature verifies and adapts it if so) and sends its output to $\mathcal{A}$.

(b) If $\deg(p_i) = 1$, define $p_i = a_0 + a_1 X_1 + \ldots + a_n X_{q+1}$. If $k_i = g^{a_0} \prod_{k=1}^{q+1} k^{a_k}$ and $\text{PreVf}(v_{ki}, m_i, k_i, c_i) = 1$, $R$ uses a query $\text{DL}(k_i)$ to get $x_j$ and outputs $\Pi_{ADP}.\text{Adapt}(\hat{\sigma}_i, x_j)$. Otherwise, it sends $\perp$ to $\mathcal{A}$.

Observe that $R$ returns $\perp$ without querying $\text{DL}(\cdot)$ for all $\perp$ $A^{2L}$-queries $\mathcal{A}$ makes. Thus it makes at most $q$ queries to $\text{DL}(\cdot)$. If $\mathcal{A}$ outputs winning values $(r_1, \ldots, r_{q+1})$, $R$ outputs the same values, thereby winning the OMDL game. By assumption, $\mathcal{A}$ succeeds with non-negligible probability, and thus $R$ also wins with non-negligible probability. This violates the OMDL assumption, implying that no such adversary $\mathcal{A}$ can exist.

**Proof of Theorem 4.9.**

**Proof.** We argue about each property separately.

**Lemma C.1 (Blindness).** Assuming $\Pi_{NIZK}$ is sound, the $A^2L^+$ scheme is blind in the LOE model.

Proof. This holds information-theoretically. Fix any two $P\text{Promise}$ executions. We now show, via a series of hybrid experiments, that the cases of $b = 0$ and $b = 1$ are statistically close.
Hybrid $\mathcal{H}_0$: Run ExpBlnd with $b = 0$.

Hybrid $\mathcal{H}_1$: In both runs of PSolver, sample $r \leftarrow \mathbb{Z}_q$ and set $Y'' := g^r$ and $c'' \leftarrow \Pi_2(pk_{\mathcal{H}_1}, r)$.

Hybrid $\mathcal{H}_2$: Compute $c''$ and $Y''$ honestly using $r_1$ in the first run of PSolver and $r_0$ in the second run of PSolver.

Hybrid $\mathcal{H}_3$: Run ExpBlnd with $b = 1$.

**Claim 1.** For all PPT adversaries $\mathcal{A}$,

$$\text{EXEC} \mathcal{H}_0, \mathcal{A} \approx \text{EXEC} \mathcal{H}_1, \mathcal{A}$$

**Proof.** $Y''$ is $g$ raised to a uniform element and $c''$ is an encryption of the same uniform element in both experiments, conditioned on the ciphertext provided by the Hub being well-formed. Thus, any distinguishing advantage necessarily corresponds to a violation of the soundness property of $\Pi_{\text{NIZK}}$. It follows that the executions are statistically indistinguishable. □

**Claim 2.** For all PPT adversaries $\mathcal{A}$,

$$\text{EXEC} \mathcal{H}_1, \mathcal{A} \approx \text{EXEC} \mathcal{H}_0, \mathcal{A}$$

**Proof.** This holds by the same logic as Claim 1. □

**Claim 3.** For all PPT adversaries $\mathcal{A}$,

$$\text{EXEC} \mathcal{H}_0, \mathcal{A} \equiv \text{EXEC} \mathcal{H}_0, \mathcal{A}$$

**Proof.** The change is only syntactical and the executions are identical.

Hence, the cases of $b = 0$ and $b = 1$ are statistically indistinguishable. □

**Lemma C.2 (Unlockability).** Assuming that $\Pi_{\text{ADP}}$ is witness extractable, pre-signature adaptable, and unforgeable the $A^2L^*$ scheme is unlockable.

**Proof.** We consider two cases separately.

1. Therefore, the adversary succeeds in this case with negligible probability.

$(b_2 = 1) \lor (b_1 = 1)$: In this case, the adversary is able to produce a valid signature on a message without seeing any pre-signature on it. This only happens with negligible probability by the unforgeability of the adaptor signature scheme.

**Lemma C.3 (Unforgeability).** Assuming the hardness of OMDL and that $\Pi_{\text{ADP}}$ is witness extractable and unforgeable, the $A^2L^*$ scheme is unforgeable in the LOE model.

**Proof.** We give a series of hybrid experiments, show they are indistinguishable, and prove by reduction to OMDL that no adversary exists with non-negligible advantage against the final hybrid.

Hybrid $\mathcal{H}_0$: This is the normal game $\text{ExpUnforg}$ (Figure 7).

Hybrid $\mathcal{H}_1$: Simulate all NIZK proofs using $\Pi_{\text{NIZK-Sim}}$.

Hybrid $\mathcal{H}_2$: If $\exists i \in [q]$ such that $\text{Vf}(vk_i^H, m_i, s_i) = 1$ and $(vk_i^H, \cdot) \in L$ but $(vk_i^H, m_i) \not\in L$, return 0.

Hybrid $\mathcal{H}_3$: If $\exists i \in [q]$ such that $\text{Vf}(vk_i^H, m_i, s_i) = 1$ and $g^\text{Ext}(s, s_i) = Y_i$, return 0.

**Claim 4.** For all PPT adversaries $\mathcal{A}$,

$$\text{EXEC} \mathcal{H}_0, \mathcal{A} \equiv \text{EXEC} \mathcal{H}_1, \mathcal{A}$$

**Proof.** This follows directly from zero-knowledge of $\Pi_{\text{NIZK}}$. □

**Claim 5.** For all PPT adversaries $\mathcal{A}$,

$$\text{EXEC} \mathcal{H}_1, \mathcal{A} \equiv \text{EXEC} \mathcal{H}_0, \mathcal{A}$$

**Proof.** The hybrids differ only in the case where the attacker returns a valid signature on a message that was not part of the transcript. By the unforgeability of the adaptor signature, this happens only with negligible probability. □

**Claim 6.** For all PPT adversaries $\mathcal{A}$,

$$\text{EXEC} \mathcal{H}_0, \mathcal{A} \equiv \text{EXEC} \mathcal{H}_1, \mathcal{A}$$

**Proof.** Any distinguishing advantage corresponds to the case in which $\mathcal{A}$ outputs some tuple $(vk_i^H, m_i, s_i)$ such that, for corresponding $(Y_i, \tilde{s}_i)$, $g^\text{Ext}(\tilde{s}, s_i) \neq Y_i$. In this case, we can give a reduction to witness extractability of $\Pi_{\text{ADP}}$. The reduction runs the setup as in $\mathcal{H}_3$ and receives a verification key $vk$ from the witness extractability game. It now picks some guess $i^* \leftarrow \{1, \ldots, q - 1\}$ (where $q - 1$ is the number of queries of the adversary) for the distinguishing index and starts $\mathcal{A}$ on $ek$, behaving the same way as $\mathcal{H}_3$ for all oracle queries, except for the $i^*$-th interaction, in which it sets $vk_i^H := vk$. In the execution of $\text{PPromise}$, it sends $m_i$ to the witness extractability game and receives $\tilde{s}$, which it gives to $\mathcal{A}$ instead of computing $\tilde{s}_i^H$ itself. Once $\mathcal{A}$ terminates and outputs $(vk_i^H, m_i, s_i)_{i=1}^{q-1}$, the reduction sends $s_i$ to its game. If it guessed the distinguishing index $i^*$ correctly, this is a winning signature. Suppose the distinguishing advantage is non-negligible. Since the guess is correct with probability $1/(q - 1)$, the reduction violates witness extractability also with non-negligible advantage, which is a contradiction. Hence the two experiments must be computationally close. □
Now we give a reduction from hybrid \( \mathcal{H}_5 \) to OM-CCA-A2L. Suppose there exists an adversary \( \mathcal{A} \) with non-negligible success probability in \( \mathcal{H}_5 \). We give a reduction that uses \( \mathcal{A} \) to win the OM-CCA-A2L game. The reduction is given by \( (c_1, h_1), \ldots, (c_{q+1}, h_{q+1}) \). It generates \( (ek, dk) \leftarrow \Pi_{\text{Gen}}(\gamma^3) \) and \( (vk, sk) \) as in \( \mathcal{H}_5 \) and starts \( \mathcal{A} \) on input \( ek \). For \( OP_{\text{promise}} \) queries, the reduction follows the same steps as \( \mathcal{H}_5 \) except it uses a different challenge \( h_i \) each time it generates a pre-signature. When \( \mathcal{A} \) queries \( OP_{\text{solver}} \), the reduction computes the completed signature \( \sigma_{AH} \) as the output of \( O^\mathcal{L} \) run on \( \mathcal{A} \)'s inputs \( (vk, m', Y', c', \sigma_{AH}) \). Note that since \( \mathcal{A} \) makes at most \( q \) non-\( \bot \) queries to \( OP_{\text{solver}} \), the reduction also makes at most \( q \) non-\( \bot \) queries to \( O^\mathcal{L} \), as the oracles return \( \bot \) in exactly the same cases.

Once \( \mathcal{A} \) returns \( q + 1 \) tuples \( (vk, m_j, \sigma_j) \), the reduction computes \( r_j = \Pi_{\text{Ext}}(ek_j, \sigma_j, h_j) \forall i, j \in \{q + 1\} \) until it has \( q + 1 \) non-\( \bot \) values \( r_j \) (at most \( q + 1 \) invocations of the algorithm) and outputs those values. Note that by the definition of \( \mathcal{H}_5 \), when \( \mathcal{A} \) completely successfull, \( g'' = h_i \forall i \in \{q + 1\} \). By assumption, the reduction wins the OM-CCA-A2L game with non-negligible probability. This violates OM-CCA-A2L-security of \( \Pi_{\text{ext}} \) (implied by Lemma 4.8), so no such adversary against \( \mathcal{H}_5 \) exists. Thus, no adversary with non-negligible success in \( \text{Exp}\text{NforG} \) can exist either. □

The theorem follows directly from Lemmas C.1 to C.3. □

**Proof of Theorem 5.3.**

**Proof.** We proceed by describing the UC simulator and arguing about indistinguishability from the real execution of the protocol. We consider the cases where the adversary corrupts a different subset of parties separately. We describe the simulator for a single session and the security of the overall interaction is established via a standard hybrid argument.

**H Corrupted.** We first give a simulator \( \mathcal{S}_H \), then give a series of hybrid experiments that gradually change the real experiment (i.e., the construction in Figures 10 and 11) into the ideal experiment given by the interaction of the corrupted \( H \) and the simulator \( \mathcal{S}_H \), which has access to \( \mathcal{F}_{\text{BCS}} \).

**Simulator \( \mathcal{S}_H \):** Upon receiving (\text{promise-req}, \text{B}) from \( \mathcal{F}_{\text{BCS}} \), \( \mathcal{S}_H \) proceeds as follows:

1. Ask the attacker to initiate a session and receive in return \( (Y, c, \pi_s, \sigma_{H'}^H) \). If \( \Pi_{\text{ADP}}.\text{PreVf}(vk^H_{m_{H'}, Y', \sigma_{H'}^H}) = 1 \) and \( \text{NIZK}.V((ek_{H'}, Y, c), \pi_s) = 1 \), proceed as in the protocol and send (\text{promise-res}, \gamma) to \( \mathcal{F}_{\text{BCS}} \). Otherwise, abort and send (\text{promise-res}, \bot).

2. Receive (\text{promise}, \text{pid}) from \( \mathcal{F}_{\text{BCS}} \).

3. Upon receiving (\text{solve-req}, \text{A}, \text{pid'}) from \( \mathcal{F}_{\text{BCS}} \) at some later point, sample a uniform element \( y'' \rightarrow \mathbb{Z}_q \) and generate keys \( (vk_{m', Y'}, \sigma_{AH}^{'}) \leftarrow \Pi_{\text{ADP}}.\text{Gen}(\gamma) \). Compute \( Y' = g^{y''}, \sigma_{AH}^{'}, \Pi_{\text{ADP}}.\text{PreSig}(sk_{m', Y'} \sigma_{H'}^H) \) and send them to the attacker.

4. When the attacker initiates the 2PC, run the 2PC simulator to recover its input \( dk_{m'} \). If \( ek_{m'} \neq \Pi_{\text{Gen}}(dk_{m'}) \), program the output of the 2PC to \( \bot \), otherwise to \( y' \).

5. Receive \( \sigma_{AH}^{'}, r \) in response from the attacker and check that \( \Pi_{\text{ADP}}.\text{Vf}(vk_{m', Y'} \sigma_{AH}^{'}, \sigma_{AH}^{'}, \sigma_{AH}^{'}) = 1 \). Additionally check if \( \Pi_{\text{ADP}}.\text{Ext}(\sigma_{AH}^{'}, \sigma_{AH}^{'}, Y') = \gamma' \). If both checks pass, send (\text{solve-res}, \gamma) to \( \mathcal{F}_{\text{BCS}} \). Compute \( s = \Pi_{\text{Dec}}(dk_{m'}, c) \), and output \( (\sigma_{AH}, s) \) as in the protocol; otherwise, send (\text{solve-res}, \bot) and abort.

6. If, at any point before the successful completion of step 4, the attacker produces a valid signature \( \sigma_{AH}^{'}, r \) or at any point in the protocol (including after step 4), a valid signature on a message \( m_{AH}' \neq m_{AH} \), send (\text{solve-res}, \bot) to \( \mathcal{F}_{\text{BCS}} \) and abort.

**Hybrid \( \mathcal{H}_6 \):** This corresponds to the real protocol (Figures 10 and 11).

**Hybrid \( \mathcal{H}_1 \):** Simulate the 2PC (Fig. 11, line 6) and send the output \( z \) to \( H \).

**Hybrid \( \mathcal{H}_2 \):** Replace \( Y' \) with \( Y'' = g^{y'} \) where \( y' \rightarrow \mathbb{Z}_q \). If \( \Pi_{\text{Gen}}.\Pi_{\text{gen}}(dk_{m'}) = ek_{m'} \), send \( y' \) to \( H \) instead of \( z \); otherwise, send 2PC.

**Hybrid \( \mathcal{H}_3 \):** Abort if \( z' = y' \) (after line 12 of Fig. 11).

**Hybrid \( \mathcal{H}_4 \):** Abort if any valid signature \( \sigma_{AH}^{'}, r \) is received on a different message \( m_{AH}' \neq m_{AH} \) or on any message before the 2PC has successfully completed.

**Claim 7.** For all PPT distinguishers \( \mathcal{E}, \)

\[
\text{EXEC}_{\mathcal{H}_6, A, \mathcal{E}} \approx \text{EXEC}_{\mathcal{H}_5, A, \mathcal{E}}
\]

**Proof.** This follows directly from the security of the 2PC protocol.

**Claim 8.** For all PPT distinguishers \( \mathcal{E}, \)

\[
\text{EXEC}_{\mathcal{H}_6, A, \mathcal{E}} \approx \text{EXEC}_{\mathcal{H}_5, A, \mathcal{E}}
\]

**Proof.** By the uniqueness of the decryption key and correctness of \( \Pi_{\text{dec}}, ek_{m} = \Pi_{\text{gen}}(dk_{m}) \) implies \( \Pi_{\text{dec}}(dk_{m}, \Pi_{\text{enc}}(ek_{m}, m)) = m \) for all \( m \) in the message space. Thus, the output of the 2PC is necessarily \( s + r \), where \( s \in \mathbb{Z}_q \) such that \( c = \Pi_{\text{enc}}(ek_{m}, s) \land Y = g^{y'} \) (this is guaranteed by the NIZK). Since \( r \) is uniformly random, \( y' \) is identically distributed to \( z = s + r \). The same holds for \( Y'' \) and \( Y' = g^{y'} \). Furthermore, it still holds that \( y' \) is the discrete logarithm of \( Y'' \) (cf. \( z \) and \( Y' \)).

**Claim 9.** For all PPT distinguishers \( \mathcal{E}, \)

\[
\text{EXEC}_{\mathcal{H}_6, A, \mathcal{E}} \approx \text{EXEC}_{\mathcal{H}_5, A, \mathcal{E}}
\]

**Proof.** If \( z' = y' \), by the uniqueness of dlog witnesses \( g^{y'} = Y'' \). By the witness extractability of \( \Pi_{\text{ADP}}, Pr[y' = Y'' \land \text{ADP}.\text{Vf}(vk_{AH}^{'}, \sigma_{AH}^{'}, \sigma_{AH}^{'}) = 1] \) is negligible, so the abort only happens with negligible probability.

**Claim 10.** For all PPT distinguishers \( \mathcal{E}, \)

\[
\text{EXEC}_{\mathcal{H}_6, A, \mathcal{E}} \approx \text{EXEC}_{\mathcal{H}_5, A, \mathcal{E}}
\]

**Proof.** Any distinguishing advantage implies a case in which \( \mathcal{A} \) outputs some valid signature \( \sigma_{AH}^{'}, r \) for some message \( m_{AH}' \) for which it has potentially been given a presignature \( \sigma_{AH}^{'} \) and corresponding statement \( Y \). This signature is a winning instance in the unforgery experiment for \( \Pi_{\text{ADP}} \), but by assumption this only occurs with negligible probability, and so the distinguishing advantage must be negligible. Therefore the experiments are statistically close.
Claim 11. For all PPT distinguishers $E$,
\[ \text{EXEC}_{H, \mathcal{A}, E} \equiv \text{EXEC}_{\mathcal{F}_{\text{BCS}}, S, E} \]

Proof. $\mathcal{H}_4$ is identical to the ideal world. \hfill \Box

A,B Corrupted. Again, we give a simulator $\mathcal{S}_{AB}$ that interacts with $\mathcal{F}_{\text{BCS}}$ and show by a series of hybrids that our protocol is indistinguishable from ideal experiment in which the corrupted parties interact with the simulator $\mathcal{S}_{AB}$.

Simulator $\mathcal{S}_{AB}$: When a recipient Bob indicates he would like to initiate a transaction, $\mathcal{S}_{AB}$ proceeds as follows:

1. Send (PPromise, $A$) to $\mathcal{F}_{\text{BCS}}$.
2. Upon receiving (promise, (pid, $p$')) from $\mathcal{F}_{\text{BCS}}$, sample a uniform value $s \leftarrow \mathbb{Z}_q$ and compute $Y \leftarrow g^s$.
3. Generate keys $(ek_H, dk_H) \leftarrow \Pi_H.KGen(1^n)$ and $(vk_H, sk_H) \leftarrow \Pi_{\text{ADP}}.KGen(1^n)$; let $c \leftarrow \Pi_E.Enc(ek_H, 0)$ and $\delta_H \leftarrow \Pi_{\text{ADP}}.\text{PreSig}(sk_H, m_H, Y)$. Simulate the NIZK $\pi_5$ to $\Pi_{\text{ADP}}.\text{Sim}(td, (ek_H, Y, c))$. Finally, pre-compute $\delta_H \leftarrow \Pi_{\text{ADP}}.\text{Adapt}(\delta_H, s)$ and save $(\text{pid, } p', (Y, c, s, \delta_H, \bot))$ into a table $\mathcal{P}$. Send $(Y, c, s, \delta_H)$ to the attacker (who is impersonating Bob).
4. At a later point in time, the attacker sends $(Y', \delta_H')$ on behalf of Alice. If $\Pi_{\text{ADP}}.\text{PreVf}(vk_H, m_{AH}, Y', \delta_H') \neq 1$, abort.
5. When the attacker initiates the 2PC, run the 2PC simulator to recover its inputs $(c', r^*)$; compute the result $(\bot)$ and return it to the attacker.

(5) Depending on whether or not $c^* \in \mathcal{P}$ do the following:

(a) If $c^* \in \mathcal{P}$, retrieve the corresponding $Y$, $s$, and $p'$. Check that $Y' = Y \cdot g^s$ (if not, abort); send $\Pi_{\text{ADP}}.\text{Adapt}(\delta_{AH}, s + r^*)$ to the attacker masquerading as Alice and $(\text{PSolver}, B, p')$ to $\mathcal{F}_{\text{BCS}}$. Update the last element of the entry in $\mathcal{P}$ to $\top$.

(b) If $c^* \notin \mathcal{P}$, compute $z' \leftarrow \Pi_E.\text{Dec}(dk_H, c^*) + r^*$. Check that $Y' \equiv g^{z'}$ (if not, abort) and send $\Pi_{\text{ADP}}.\text{Adapt}(\delta_{AH}, z')$ to the attacker. Send nothing to $\mathcal{F}_{\text{BCS}}$. (Note that this corresponds to the case where some party Alice is paying Hub without Bob initiating the interaction, which is something that she can do at any time.)

(6) When the attacker outputs some valid signature $\sigma_{AB}^H$, check that the following conditions hold: $\Pi_{\text{ADP}}.\text{PreVf}(\text{Vf}(vk_H, m_{AH}, \sigma_{AB}^H)) = 1$ and $((\text{pid, } p', (\bot, c, \delta_{AB}^H, \top)) \in \mathcal{P})$. If so, send (Open, pid) to $\mathcal{F}_{\text{BCS}}$; otherwise, abort.

Hybrid $H_6$: This corresponds to the real protocol (Figures 10 and 11).

Hybrid $H_7$: Replace the honestly-computed NIZK $\pi_5$ (Figure 10, line 4) with a simulated proof.

Hybrid $H_8$: Simulate the 2PC (Figure 11, line 6).

Hybrid $H_9$: Add the list $\mathcal{P}$ and step 5 of the simulator (in particular, case 5a) to Figure 11, line 7-10.

Hybrid $H_{10}$: Replace $c$ (Figure 10, line 2) with an encryption of zero.

Hybrid $H_{11}$: When Bob outputs a valid signature, abort if $(\bot, (\bot, c, \delta_{AB}^H), b) \in \mathcal{P}$ and $b \neq \top$.

Claim 12. For all PPT distinguishers $E$,
\[ \text{EXEC}_{H, \mathcal{A}, E} \equiv \text{EXEC}_{H_6, \mathcal{A}, E} \]

Proof. This follows directly from the zero-knowledge property of the NIZK. \hfill \Box

Claim 13. For all PPT distinguishers $E$,
\[ \text{EXEC}_{H, \mathcal{A}, E} \equiv \text{EXEC}_{H_5, \mathcal{A}, E} \]

Proof. This follows directly from the UC-security of the 2PC protocol. \hfill \Box

Claim 14. For all PPT distinguishers $E$,
\[ \text{EXEC}_{H, \mathcal{A}, E} \equiv \text{EXEC}_{H_5, \mathcal{A}, E} \]

Proof. By definition, for $c^* \in \mathcal{P}$, the corresponding $s$ and $Y$ in $\mathcal{P}$ are $\Pi_E.\text{Dec}(dk_H, c^*)$ and $g^s$, respectively. Therefore $z' = s + r^*$ and the case of $c^* \in \mathcal{P}$ is handled in the same way as all cases were in the previous hybrid experiment. \hfill \Box

Claim 15. For all PPT distinguishers $E$,
\[ \text{EXEC}_{H, \mathcal{A}, E} \equiv \text{EXEC}_{H_5, \mathcal{A}, E} \]

Proof. Suppose towards a contradiction that $E$ can distinguish the two executions with nonnegligible probability. We give a reduction to the CCA-security game of $\Pi_E$. The reduction sets $m_0 := s$ and $m_1 := 0$, sends them to the CCA game, and receives $c$. It then acts as hub in its interaction with $E$, computing everything as in Hybrid 3, except for $c$, which it sets to the ciphertext it received from the game. When it needs to decrypt $c^*$ it uses the CCA decryption oracle. At the end of the execution, based on $E$’s guess, it outputs a bit to the CCA game (0 if $E$ guesses $H_5$, 1 otherwise), which will be correct with nonnegligible advantage. This violates the CCA-security of $\Pi_E$, so the two executions must be indistinguishable. \hfill \Box

Claim 16. For all PPT distinguishers $E$,
\[ \text{EXEC}_{H, \mathcal{A}, E} \equiv \text{EXEC}_{H_6, \mathcal{A}, E} \]

Proof. If $b \neq \top$, Alice did not receive the completed signature $\sigma_{AH}^H$ for that session and thus cannot recover the secret $s$ to send to Bob. This means Bob’s signature $\sigma_{HB}^H$ was created without knowing the witness for the pre-signature $\sigma_{AB}^H$, which, by $\text{ADP}$ of $\Pi_{\text{ADP}}$, can only happen with negligible probability. Thus the abort also only happens with negligible probability and the two experiments are indistinguishable. \hfill \Box

Claim 17. For all PPT distinguishers $E$,
\[ \text{EXEC}_{H, \mathcal{A}, E} \equiv \text{EXEC}_{\mathcal{F}_{\text{BCS}}, S, E} \]

Proof. $\mathcal{H}_5$ is identical to the ideal world. \hfill \Box

A,H Corrupted. This case is trivial, as $\mathcal{H}$ has no secret information and the simulator therefore simply follows the protocol.

H,B Corrupted. The simulator in this case follows the protocol honestly. If Hub publishes a valid signature $\sigma_{AH}^H$ on a transaction $m$ that is not in the simulator’s (acting as Alice) transcript, the simulator aborts. This means that the adversary was able to forge a signature $\sigma_{AH}^H$ on some transaction $m$ for which it did not previously receive a pre-signature $\sigma_{AB}^H$. By $\text{EUF-CMA}$ of the adaptor signature scheme, this case occurs with negligible probability and thus for all PPT distinguishers $E$, the real world (an honest execution of the
<table>
<thead>
<tr>
<th>Ideal Functionality $\mathcal{F}_{\text{BCS}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Puzzle Promise:</strong> On input $(\text{PPromise}, A)$ from $B$, $\mathcal{F}_{\text{BCS}}$ proceeds as follows:</td>
</tr>
<tr>
<td>- Send $(\text{promise–req}, B)$ to $H$ and $S$.</td>
</tr>
<tr>
<td>- Receive $(\text{promise–res}, b)$ from $H$.</td>
</tr>
<tr>
<td>- If $b = \bot$ then abort.</td>
</tr>
<tr>
<td>- Sample $\text{pid}, \text{pid}' \leftarrow {0, 1}^n$.</td>
</tr>
<tr>
<td>- Store the tuple $(\text{pid}, \text{pid}', \bot)$ into $P$.</td>
</tr>
<tr>
<td>- Send $(\text{promise}, (\text{pid}, \text{pid}'))$ to $B$, $(\text{promise}, \text{pid})$ to $H$, $(\text{promise}, \text{pid}')$ to $A$, and inform $S$.</td>
</tr>
</tbody>
</table>

| **Puzzle Solver:** On input $(\text{PSolver}, B, \text{pid}')$ from $A$, $\mathcal{F}_{\text{BCS}}$ proceeds as follows: |
| - If $(\cdot, \text{pid}', \cdot) \notin P$ then abort. |
| - Send $(\text{solve–req}, A, \text{pid}')$ to $H$ and $S$. |
| - Receive $(\text{solve–res}, b)$ from $H$. |
| - If $b = \bot$ then abort. |
| - Update entry to $(\cdot, \text{pid}', \top)$ in $P$. |
| - Send $(\text{solved}, \text{pid}', \top)$ to $A, B$ and $S$. |

| **Open:** On input $(\text{Open}, \text{pid})$ from $B$, $\mathcal{F}_{\text{BCS}}$ proceeds as follows: |
| - If $(\text{pid}, \cdot, b) \notin P$ or $b = \bot$ then send $(\text{open}, \text{pid}, \bot)$ to $B$ and abort. |
| - Else send $(\text{open}, \text{pid}, \top)$ to $B$. |

Figure 15: Ideal functionality $\mathcal{F}_{\text{BCS}}$ (corresponds to $\mathcal{F}_{\text{A2L}}$ in [53]). Portions related to grieving protection (i.e., registration) have been removed.