How to Recover a Cryptographic Secret From the Cloud

Chris Orsini, Alessandra Scafuro[*], and Tanner Verber[†]

North Carolina State University

September 1, 2023

Abstract

Clouds have replaced local backup systems due to their stronger reliability and availability guarantees compared to local machines, which are prone to hardware/software failure or can be stolen or lost, especially in the case of portable devices

In recent years, some digital assets are managed solely through the knowledge of cryptographic secrets (e.g., cryptocurrency, encrypted datasets), whose loss results in the permanent loss of the digital asset. Since the security of such systems relies on the assumption that the cryptographic key remains secret, a secret owner Alice cannot simply store a backup copy of such secret on the cloud, since this corresponds to giving away her ownership over the digital assets. Thus Alice must rely on her personal machines to maintain these secrets.

Is it possible to obtain the best of the two worlds, where Alice benefits from the convenience of storing a backup copy of her cryptographic secrets on the cloud such that she can recover them even when she loses her devices and forgets all credentials, while at the same time retaining full ownership of her secrets?

In this paper, we show that this is indeed possible, by revisiting and expanding the concept of Break-glass Encryption pioneered by Scafuro [PKC19].

We provide a secret-recovery mechanism where confidentiality is always guaranteed when Alice has not lost her credentials, even in the presence of a malicious cloud and users ([PKC19] only guarantees that a violation of confidentiality will be *detected*, not prevented). Recoverability is achieved in most circumstances.

We design and prove security of a credential-less authentication mechanism, that enables Alice to access her secret, without remembering any credentials. This tool was assumed in [PKC19] but not implemented. We redesign the storage mechanism on the cloud side so that the cloud needs to perform *no operations* during the storage phase. This is in contrast with [PKC19] where the cloud must re-encrypt the stored file continuously with the help of a secure enclave (regardless of whether a recovery procedure will happen).

Our protocols are proved secure in the Universal Composition framework.

1 Introduction

Some digital assets, such as cryptocurrencies [MSH+16] or encrypted databases [PRZB12], require the knowledge of a cryptographic secret (e.g., a signing key, a decryption key,) in order to be accessed and used by its owner. The security of such digital assets hinges on the cryptographic secret being known *only* by its owner, and, if the owner loses the secret, she loses access to this asset[1].

To avoid this loss, could a user, say Alice, store a backup copy of her cryptographic secrets on a cloud, while at the same time retaining full ownership of her secrets? It seems that the answer to this question should be a definitive no. Firstly, by sending the cryptographic secrets s to the cloud, the user is effectively giving the cloud the ability to access the cryptographic asset associated with s, defeating the purpose of

[*]Alessandra Scafuro and Tanner Verber are supported by a research grant from Horizen Labs

[†]Author contact tverber@ncsu.edu.

[1]This is a severe problem in cryptocurrencies where losing access to the signing key results in losing the ability to create transactions and hence use the money.

https://t.me/learningnets

using a cryptographically secured digital asset to begin with. However, even if we trust the cloud not to illegitimately access users' assets, another potential point of failure is the authentication method that Alice uses to connect to the cloud. The security of the cryptographic systems for which Alice is using her cryptographic secrets is downgraded to the security of the authentication system Alice uses with the cloud. Finally, if Alice uses a strong two-factor authentication (2FA) involving her physical devices as second factors, there is still a chance that Alice loses access if she loses all her physical devices.

In this paper, we revisit the concept of Break-glass Encryption, introduced by Scafuro [Sca19], to provide an efficient and concrete mechanism that allows Alice to use the convenience of a cloud to store a cryptographic secret, in such a way that (1) the cloud does not learn the secret (2) Alice is able to retrieve her secret even when she loses all her credentials and devices, while no-one else can retrieve the secret on her behalf.

Such a mechanism seems to have two contradicting security requirements. On the one hand, we must provide provable cryptographic guarantees that no one, not even the cloud, should be able to learn the cryptographic secret that Alice is storing. On the other hand, we must guarantee that if Alice loses all her devices and secrets – which puts Alice in a position of being like anyone else – she should be able to connect to the cloud and retrieve her secret. While seemingly irreconcilable, we show that this can be achieved.

In order to better frame the ideas behind our scheme, we present the overarching approach we take in this paper to achieve the security requirements outlined above. The first requirement asks that Alice stores her secret s on the cloud, in such a way that the cloud will never learn the secret. This would be easily achieved by having Alice store an encryption of s, under some key that Alice knows, say retK, that we will call "retrieval key". As long as Alice remembers this key, she will be able to retrieve and decrypt her cryptographic secret. The second requirement asks that if Alice loses *everything*, Alice should be able to retrieve and decrypt s. This is impossible unless Alice gave her retrieval retK to someone else, but we want that no one should be able to access this secret besides Alice. Note that this requirement rules out any solution based on escrowing the key to trusted parties [Mic92, Sha95, BG96, Gan96, BG97]. In particular, we want that no one should ever be able to access Alice's secret without her detecting it (which would happen if the key is escrowed to others).

To overcome this challenge we use the aid of a trusted execution environment (TEE) [MAB+13, PST17]. TEE is a technology that allows a client to create her own private space on an untrusted cloud machine (often called the host). The assumption is that every computation and piece of data stored in the enclave is opaque to the host.[2] Hence, we require that the cloud is equipped with a TEE and Alice can create her own secure enclave to privately store retK. The TEE can be seen as the trusted party that knows about Alice's key retK and can use it in case of emergency to decrypt Alice's ciphertext, recover her secret s, and *re-encrypt* it under a new key, say pk, that Alice freshly created. This is done obliviously to the cloud who just observes the ciphertext c as input to the enclave, and then the fresh ciphertext c' in output, which is a re-encryption of c under a new key pk.

However, the most challenging question remains unanswered. How does the TEE know that pk is a key chosen by Alice, and not by the cloud? Or another party who is pretending to be Alice?

This brings us to the second, and most challenging, tool that we build in this paper: a mechanism to obtain credential-less cryptographically secured permission. This mechanism, inspired by a similar definition presented, but not implemented, by Scafuro [Sca19] allows *only* Alice to legitimately authenticate, even when she loses all cryptographic secrets. This permission should be publicly verifiable and can be inputted into the TEE to authenticate a recovery request from Alice. This again seems very improbable since any mechanism that would allow Alice to obtain a certificate without any particular secret, should also enable any party to do the same.

However, we observe that there is a difference between Alice and the others. Alice would ask for a certificate only in the rare case that she lost all the devices/credentials she possesses. Most of the time, Alice does have such credentials – and can use them to authenticate. In contrast, other parties are never able to authenticate.

This asymmetry can be leveraged to create a permission mechanism that guarantees that, using her keys, Alice can stop any malicious requests for permission, and that in the case where Alice lost everything, her attempt to create permission cannot be (cryptographically) stopped by anyone. Our solution requires

[2] In practice, TEEs are still not bulletproof since hardware side-channel attacks exist. Nevertheless, this technology is still developing and it is used in several works [DSC+15, KMB15, ZDB+17, KGM19, CZK+19]

https://t.me/learningnets

that Alice uses her credentials (as long as she has them) to actively participate in stopping illegitimate attempts at recovery, as well as initiate a recovery request as soon as she realizes that she lost them. The latter is necessary, else, if Alice loses her credentials and does not initiate a recovery request, she loses her power to stop any future illegitimate requests. Looking ahead, in our solution, there is a possibility of a non-cryptographic attack that prevents Alice, whereby an adversary *guesses* that Alice will make a recovery request at a specific time, and will initiate a recovery request in parallel. As we discuss later, this attack would prevent Alice from recovery but will never compromise the security of Alice's secrets since two competing requests will only result in aborting the recovery process. Likewise, if Alice fails to reject a malicious request, she will lose her confidentiality. Therefore, Alice needs to ensure that she watches for any requests made on her behalf.[3]

Our Contribution. We build upon the ideas presented in [Sca19] to provide a full protocol that allows clients to securely recover cryptographic secrets stored on a cloud even without remembering any credentials while guaranteeing that no one else, not even the cloud, learns the secret. Our protocol requires the implementation of two tools: a tool to cryptographically authenticate a client when she loses all her credentials and a tool to allow the cloud to obliviously recover a secret for the client, without learning the secret. We defined the two tools in a modular way and we implement them independently. We provide the following contributions:

1. **The first implementation of an authenticated Credentials-less Permission Mechanism $\mathcal{G}_{Perm}$ using blockchains.** While [Sca19] abstracted the concept of a credentials-less permission mechanism via the $\mathcal{G}_{Perm}$ functionality, it was never instantiated. We provide the first concrete protocol that securely realizes the $\mathcal{G}_{Perm}$ functionality. In proving UC-security of such a protocol many subtleties arise that highlight issues with previous definitions and proposed instantiations. We therefore revisited and improved the definition of $\mathcal{G}_{Perm}$. Our implementation leverages blockchain technology.
2. **UC-definition of a credential-less Secret Recovery functionality $\mathcal{F}_{SecRec}$.** We provide a formal definition of a secure yet credential-less Secret-Recovery mechanism, in the UC-framework.
3. **UC-protocol instantiating $\mathcal{F}_{SecRec}$ in the $\mathcal{G}_{Perm}$ hybrid world, and using TEE.** We instantiate $\mathcal{F}_{SecRec}$ with a very efficient protocol that leverages the security of TEE, and requires minimal overhead for the cloud. We provide a formal UC-security proof of our protocol, and hence we use the UC-formalization of TEE provided by [PST17], called $\mathcal{G}_{att}$. Our protocol is modular and uses $\mathcal{G}_{Perm}$ as a module that can be instantiated with other implementations besides the one we provide in this paper.

Roadmap In Section 2 we give a high level description of our techniques to achieve our solution. Section 3 discusses our improvements over Break-glass Encryption [Sca19] and other related works. Section 4 gives background on the tools used in our solution. In Section 5 gives our definition, protocol, and proof for credential-less publicly verifiable permissions. Finally, Section 6 gives our definition, protocol, and proof for credential-less secret recovery.

2 Our Techniques

1. Implementation of credential-less permission mechanism $\mathcal{G}_{Perm}$ via blockchain. We revisit the definition of credential-less permission provided by Scafuro [Sca19] in our own ideal functionality $\mathcal{G}_{Perm}$ (Figure 7). Our definition includes technical changes to improve modularity. Among these changes, our formulation defines the role of clients and servers, and their connection, and better defines the role of external parties in requesting permission, and the verification of the permission is defined as an external predicate. These changes make $\mathcal{G}_{Perm}$ usable as a building block in any application that requires authenticated credential-less permission. Our main contribution on this front, however, is in our *UC-realization of* $\mathcal{G}_{Perm}$. While Scafuro provided a definition and a discussion of potential realizations, this work is the first to concretely realize $\mathcal{G}_{Perm}$ and prove security.

At a high level, our blockchain-based realization is as follows. To register to the permission system, Alice posts a transaction indicating the identifier she wants to use $\mathsf{perm\text{-}info}$, a public key $\mathsf{vk_C}$ that she wants to use to block/accept permission requests, and the server(s) that are allowed to use her permissions.

[3] As we will see, this watching can be outsourced.

https://t.me/learningnets

Only servers that have published their verification key to the blockchain can be chosen by Alice. In this registration transaction, Alice also establishes timing parameters related to the creation of the permission: t_{open} and t_{chal}, which will become clear later in the description. In order to have a transaction posted on the blockchain Alice might need to create a blockchain account (e.g., a wallet). However, note that for our protocol Alice does not need to remember the secret key associated to it after the transaction is computed. Indeed, Alice can create an account on the fly for each transaction that she wishes to post. Also, note that in our current version of the protocol we do not consider the expenses of posting transactions. However, they could be leveraged to discourage malicious parties from posting illegitimate requests and to reward Alice's attempt to stop them (we discuss this more in the *Competing Requests* paragraph of Section 5.2).

Now, say Alice loses the keys she used to access to the server S. She will create a permission that S can publicly verify as follows. First, she posts a transaction tx on the blockchain containing her public id $\mathsf{perm\text{-}info}$ and the public key of the server S she wants to create permission for and a field that we call req, which contains additional information for the server S. (This field is specific to the application for which the permission is used. Looking ahead in Secret Recovery req will be a fresh public key pk that the TEE will use to re-encrypt the secret it holds for Alice). Once Alice's transaction appears on the blockchain, there are two cases. If Alice does remember the signing key associated with $\mathsf{vk_C}$ that was registered along with $\mathsf{perm\text{-}info}$, Alice can simply endorse by signing this transaction.[4] If Alice does not remember any key, then Alice simply waits for t_{chal} blocks to be added to the ledger after the block containing the transaction tx is posted. This sequence of blocks, tx and the t_{chal} blocks after, simply represents a valid permission for S. We call this a *silent proof*. Note that constructing this sequence required no secrets from Alice.

On the other hand, assume that Alice did not lose her secrets, and a malicious party, Jeff, attempts to create permission by following the procedure above. Jeff can simply create a transaction tx^* containing $\mathsf{perm\text{-}info}$ and then wait for t_{chal} blocks to be appended. The key observation here is that, if Alice did not lose her devices, then she has access to the signing key associated with $\mathsf{vk_C}$, so she can post a signature to deny the request and stop anyone else from obtaining a silent proof. In this case, when Alice sees the transaction tx^*, she will follow up with a transaction where she denies tx^* and add a signature that verifies under $\mathsf{vk_C}$. The ability to deny is why we consider silence to be proof of accepting permission.

Note that this requires that Alice monitors the blockchain and is on the lookout for illegitimate permission requests. While this might seem too taxing for Alice, monitoring the ledger could be offloaded to a server who notifies Alice in the case of a request. Further, the following observations show that this work can move from taxing to rewarding. First, the parameter t_{chal} plays an important role in the frequency of the monitoring activity. Alice could choose t_{chal} to be long enough (e.g., one week) so that she does not have to monitor the blockchain constantly, but has a monitoring procedure going off once a week. Second, while we do not explicitly implement this in the paper, every permission request can have a required request fee that is automatically paid to the wallet associated with $\mathsf{perm\text{-}info}$ when denied. In this way, every denial yields Alice a reward.

Now, consider the last, most challenging case, where Alice loses her keys and posts a transaction tx requesting permission for $\mathsf{perm\text{-}info}$, S, req, and this is immediately followed by a transaction from another party tx^* for the same $\mathsf{perm\text{-}info}$, S, req'. Since Alice lost her keys, no one can stop tx^*.

This attack is similar to the "front-running attacks" that plague blockchain applications, where miners can take advantage of their first-hand knowledge of the transactions in the mempool and create transactions accordingly (e.g., if the transactions suggest a certain buying trend, the miners can take advantage of it before anyone else by creating and validating their own transactions first).

Luckily the front-running problem can be greatly mitigated using a commit-and-reveal approach where parties do not post values in the clear, but rather they post a commitment to it. Once the transaction containing the commitment made it to the blockchain, the party can follow up with a transaction that contains the opening of the commitment. Thanks to the hiding of the commitment, when Alice publishes the commitment to $\mathsf{perm\text{-}info}, \mathsf{S}$, and req, Jeff will not be able to know what $\mathsf{perm\text{-}info}$ is committed and will not be able to front-run Alice. The commit-and-reveal approach increases the overhead on the blockchain and the users, however, for our setting this is not problematic since permission transactions are expected to be infrequent. Hence, the protocol discussed above is slightly modified so that permission request transactions

[4]This step might seem redundant but it will become clear later why we break it down into two transactions.

https://t.me/learningnets

are first committed and then opened. A parameter t_{open} is then used to establish how many blocks can pass for an opening to be accepted by our system.

However, this does not entirely stop an adversary from determining the identity of Alice. A malicious miner monitoring network traffic could determine the source of a commitment, giving a strong indication of Alice's identity. To avoid this, an anonymous network may be used such as Tor [Tora] or, if the ledger is a bitcoin-like ledger, Dandelion [VFV17].

A Non-Cryptographic Attack. Note that although hiding guarantees that Jeff cannot detect which perm-info is committed in the permission request, there is still a possibility that Jeff tries to mount a denial of service attack to the system by publishing commitments to all perm-infos, with the hope of guessing the one that is committed in an honest transaction. Furthermore, Jeff might be someone that knows that Alice was robbed of their phone and could be the one posting a transaction perm-info right before (or right after) Alice posts her request.

All such attacks are not cryptographic in nature, as they concern the ability of Jeff to predict which perm-info will be lost. The consequence of this is that two transactions with the same perm-info will appear on the blockchain. In this case, our protocol will simply ignore the request and no permission will be created. Note that this approach guarantees that in case of doubts, no one gets permission, which means that Alice's secrets remain protected (although they are not recoverable by Alice).

2. UC-definition of a Credential-less Secret Recovery Functionality $\mathcal{F}_{SecRec}$. We provide a UC-definition of the properties we want from a cloud-assisted secret recovery functionality where the cloud does not learn anything, besides the fact that a client wishes to store a secret.

We model this via the ideal functionality $\mathcal{F}_{SecRec}$ (Figure 21), where a client Client can store a secret s with $\mathcal{F}_{SecRec}$ and a cloud Cloud is informed that a client Client has stored a secret and has a public identity perm-info associated to it. The client Client can ask the ideal functionality to retrieve her secret s anytime. Upon this request, $\mathcal{F}_{SecRec}$, before sending s to Client, will first need the approval from Cloud. This step models the fact that in the real world a cloud can always stop working or refuse to provide service.

Any party P can ask for a recovery request. This is an emergency request that may come from someone other than the owner. If this request is associated with a valid permission perm, $\mathcal{F}_{SecRec}$ accepts and sends the secret s to this unauthenticated party P – again assuming that Cloud has agreed to provide this service. Thanks to the modular UC-definition perm can be checked in $\mathcal{F}_{SecRec}$ by accessing $\mathcal{G}_{Perm}$.

Finally, we also allow a client to remove her secrets from the cloud.

3. UC-protocol instantiating $\mathcal{F}_{SecRec}$ in the $\mathcal{G}_{Perm}$ hybrid world, and using TEE. We provide a realization of $\mathcal{F}_{SecRec}$ in the $\mathcal{G}_{Perm}$ hybrid world and using the TEE, modeled as an ideal functionality $\mathcal{G}_{att}$ [PST17]. In the realization, the first step for Alice is to register with $\mathcal{G}_{Perm}$ ideal functionality, choosing a public id perm-info and communicating the identity of the cloud Cloud she wants to associate the permission to.

Then, to store a secret s on Cloud's machine, as mentioned above, Alice interacts with the TEE hosted by the cloud (in our formal protocol, Alice interacts with the $\mathcal{G}_{att}$ functionality). The TEE executes a simple program. (1) perform key-agreement with a client with id perm-info, (2) process recovery requests for the client if they are authenticated with a valid permission perm.

Alice engages in a Diffie-Hellman key-agreement protocol[5] with the TEE hosted at the cloud, we denote this key as "retrieval key" retK. This key is used by Alice to encrypt s, via a CCA-secure encryption scheme, and obtain the ciphertext c, which is then stored on the cloud Cloud. So long as Alice remembers her retrieval key retK, Alice can retrieve her secret by simply downloading c and decrypting it with retK.

If Alice loses her key(s), she will use $\mathcal{G}_{Perm}$ to obtain permission perm. $\mathcal{G}_{Perm}$ allows a party to create permission for a specific action req. In our protocol, the action is to have the TEE recover the secret and re-encrypt it under a new public key pk that Alice picks. Hence, in our protocol req = ("*recover*"||pk).

After obtaining the permission perm from $\mathcal{G}_{Perm}$, Alice sends the pair (req, perm) to the cloud which will be used to operate TEE. The TEE is queried with input (req, perm, perm-info, c) and will first check that

[5]Note that any secure key exchange would work, we chose DHKA for simplicity.

https://t.me/learningnets

perm verifies for perm-info. If the permission is valid, the TEE will attempt to decrypt c with the key retK associated to perm-info, and re-encrypt under the public key pk.

Finally, to implement removal from the system, clients simply send an authenticated removal request to the TEE to remove their secrets from the enclave.

2.1 Areas for Improvement

Before proceeding, we take a moment to discuss areas for improving the contributions presented in this work.

Avoiding Trusted Execution Environment. Of course, trust should be limited as much as possible, and therefore the use of a trusted execution environment is slightly concerning. However, through the use of two non-colluding servers, the TEE can be replaced using secure 2PC. Further, the input to the TEE can be very large, consisting of possibly many blocks of the ledger that need to be verified. Our protocol could be improved by structuring this input as a compact data structure and a proof of membership (in the case of an acceptance signature) or a proof of non-membership (in the case of a proof of silence).

Monitoring the Ledger. It might be unrealistic to assume that a client will constantly monitor the ledger for malicious requests to access their secret. This suggests that the size of the challenge window t_{chal} must be very large to allow the client to catch these malicious requests. However, we observe that the *monitoring* (and only the monitoring) of the ledger can be outsourced to one or more servers. Servers would be trusted only to *inform* the clients on time that a request recovery was published, and nothing else. This can be a reasonable approach, which requires minimal trust, and clients can always check the ledger on their own – even when they are paying the servers to do so.

Proof of Silence. A proof of silence might not be ideal, as it does not definitively prove that the client wishes for permission to be granted. It could be possible that the client did not see that the request had been posted. While it may seem safer to have the client remember something simple, such as a PIN or passphrase, our goal was to provide a way for the client to obtain permission while remembering *nothing*. Nevertheless, it may be worthwhile to implement such a measure that allows the client to obtain permission using a PIN or passphrase, and default to a proof of silence if this has been forgotten as well.

Competing Requests. In our protocol, there is the possibility that two users will claim the same identity, neither with a way to prove it. One possible approach is to accept the first request that was committed, as is done in KELP [BCC+21]. However, there is the possibility that the adversary has learned in real time that the client has lost their credentials. For example, an adversary could physically steal the client's devices. Then post a recovery request. If the client is not able to post their own request first, then their secret is lost.

We instead opt to accept neither request when two valid competing requests are made. However, this gives an adversary the ability to block the true client's request. With the implementation of a request fee[6], returned upon the granting of permission, the adversary must risk their own money in order to steal the client's secret.

With the implementation of the request fee, there is a discussion to be had about how much an adversary would reasonably risk (and therefore, how many requests an adversary would make) to potentially steal the client's secret. We leave the game theoretic analysis of this for future work.

3 Related Work

Our work is inspired by the concept of "Break-glass Encryption" introduced by Scafuro [Sca19]. Their goal is to allow a user to decrypt her own ciphertexts stored on the cloud, even when she loses her decryption key. Being the first of its kind, [Sca19] focused on a feasibility result, and some important tools (such as the permission functionality) were defined but not realized with any protocol.

[6]Discussed in Section 5.2

https://t.me/learningnets

Our work improves on [Sca19] in several ways. We provide a concrete realization of the permission functionality $\mathcal{G}_{Perm}$ and we prove its security in the UC-framework. Along the way, we revised the definition of $\mathcal{G}_{Perm}$ and improved its modularity. Our recovery protocol is very practical, requiring our cloud to only store one ciphertext and query the TEE only four times: twice upon storage, once in the case of removal, and once in case of emergency recovery. In contrast, [Sca19] provides a complex protocol that requires the cloud to continuously update the client's ciphertexts with bookkeeping information, using trusted hardware.

At a high level, our improvement over Break-glass Encryption is in the *maintenance* of ciphertexts. Specifically, our protocol requires no maintenance of ciphertexts. In Break-glass encryption, each ciphertext must be *re-encrypted every I steps.* I is a parameter set when implementing the system. Further, note that each of these re-encryptions are done on the trusted hardware. In Secret Recovery, the ciphertext is *only re-encrypted once, upon recovery.* Therefore, Break-glass Encryption requires constant maintenance of all stored ciphertexts, Secret Recovery does not. No work needs to be done besides storage, retrieval, and recovery.

Further, we discovered and addressed problems that were unmentioned in Break-glass Encryption. For example, front-running attacks and the need for computational unforgeability of the ledger. Lastly, Break-glass Encryption *does not prevent unauthorized access by the cloud.* Break-glass Encryption only allows the client to *detect* that their secret has been accessed.

The Problem of Recovery. The problem of recovery is most popular in blockchain settings, as in our previous example of a user losing the key to their cryptocurrency wallet and subsequently their funds. The work by Blackshear et al. [BCC+21] focuses on this specific problem, and provides a mechanism that allows the owner of a wallet, who forgot the key, to replace her old wallet with a fresh wallet for which she does know the key. Their approach uses time-lock commitments and smart contracts [BCC+21]. This solution, however, only applies to cryptocurrency wallets, whereas our work solves a more general problem of recovering any kind of secret stored on a cloud.

The work by Maram, Kelkar, and Eyal [MKE22] considers a related problem that they call the authentication problem. The authors consider a scenario of two parties, an honest party and an adversary, both interacting with a mechanism to prove identity. This mechanism will use some set of credentials, or other facts that the honest party should know, to verify the identity. We instead consider the case where the user has no credentials left, rather the user has lost everything, and can still recover a stored secret.

Other approaches for recovering a forgotten key are based on key-escrow. Key-escrow is an approach to key-recovery where a trusted party stores the key on behalf of the key-owner [BG96, Gan96]. The key can also be split into parts and shared among trustees of the authority so that cooperation from a threshold amount of the trustees is required [Mic92, Sha95]. Similarly, a user might escrow only part of their key, so the authority must work to obtain the entire key [BG97]. However, due to the trusted nature of the authority, the key can be used by the authority at any time. In fact, many applications of key escrow involve law enforcement playing the role of the authority and using the key for "authorized wiretaps". One interesting approach to key-escrow was shown by Green, Kaptchuk, and Van Leer where surveillance can only occur given a warrant by a judge, either prospectively or retrospectively [GKL21]. Further, key-escrow provides no detectability. The user has no means to determine if their key has been recovered by someone else.

All such approaches are incomparable to ours since a key requirement of our work is to enable Alice to recover her secret on her own without having to share her credentials with any other party. The main motivation for insisting on this requirement is that sharing credentials with other parties results in other, potentially corrupt, parties having access to the assets associated with the credentials. In the worst case, this could result in these parties stealing said assets.

Another popular approach is to lock keys behind a password, passphrase, or passcode, as these are often easier to remember than a key. For example, in Torus [torb], users provide an email for them to receive a backup passphrase to recover their wallet. Similarly, users of the Trezor Hardware Wallets are able to recover their wallets using a seed phrase of 12, 18, 20, 24, or 33 words [tre]. While seed phrases are meant to be made up of easy to remember words, it can still be a challenge to remember that many words. Rather than longer seed phrases, in SafetyPin [DCM20], users are able to recover their cloud-based mobile backups using a short PIN. This paper also provides a safeguard against brute-force guessing attacks against these PINs. Lastly, there are approaches based on password-protected secret sharing [JKKX17, JKX18]. These protocols allow a client to prove knowledge of a password to a server, without revealing said password, to obtain a

https://t.me/learningnets

previously stored secret. Furthermore, we aim to provide a route for users who have forgotten *everything*, including any password, passphrase, or passcode.

To provide support for these users who have forgotten everything, there is also the method of biometric-based recovery. One approach to using biometrics in this space is to use biometric-based encryption to encrypt the key, that way it can always be decrypted using a fingerprint upon recovery [ACAA19]. The user thus need not remember anything, however, in this solution, the encrypted key is split among a set of "stewards" who are trusted and must be online for recovery. Our solution has a single cloud, who is not trusted but does run a TEE, and we do not require costly biometric encryption, rather standard encryption.

4 Background

In this section we present the tools that our constructions are built on.

4.1 Symmetric key encryption

We revisit the definition from Chapter 3 of [KL14]. Let $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric key encryption scheme where:

- Gen is the key generation algorithm that takes as input 1^λ and outputs a key k. Concretely, $k \leftarrow\$ \mathsf{Gen}(1^\lambda)$.
- Enc is the encryption algorithm that takes as input a key k and a message $m \in \{0,1\}^*$ and outputs a ciphertext c. That is, $c \leftarrow\$ \mathsf{Enc}(k, m)$.
- Dec is the decryption algorithm that takes as input a key k and a ciphertext c and outputs a message m or an error $(\perp)$. That is $m = \mathsf{Dec}(k, c)$.

For completeness, it is required that for every $\lambda \in \mathbb{N}$, every $k \leftarrow\$ \mathsf{Gen}(1^\lambda)$, and every $m \in \{0,1\}^*$, it holds that $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$.

In Figure 1, we show the IND-CPA game presented in [KL14] for a symmetric encryption scheme as described before. This game is between a challenger and a PPT adversary $\mathcal{A}$.

$\mathsf{PrivK}^{cpa}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow\$ \mathsf{Gen}(1^\lambda)$.
2. $m_0, m_1 \leftarrow \mathcal{A}^{\mathsf{Enc}_k(\cdot)}(1^\lambda)$ s.t. $|m_0| = |m_1|$.
3. $b \leftarrow\$ \{0,1\}$, $c^* \leftarrow\$ \mathsf{Enc}_k(m_b)$.
4. $b' \leftarrow \mathcal{A}^{\mathsf{Enc}_k(\cdot)}(c^*)$.
5. If $b' = b$ output 1, else output 0.

Figure 1: IND-CPA for Π

We say an encryption scheme Π is IND-CPA secure if:

$$\Pr[\mathsf{PrivK}^{cpa}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Additionally, we provide the INT-CTX experiment [BN00, Sca19] in Figure 2 for a symmetric encryption scheme $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for reference.

We say an encryption scheme Π is INT-CTX secure if:

$$\Pr[Exp^{INT-CTX}_{\mathcal{A},\Pi}(1^\lambda) = \mathsf{win}] \leq \mathsf{negl}(\lambda).$$

4.2 Public key encryption.

Next we discuss the definition from Chapter 11 of [KL14]. Let $\Pi_{pub} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme and M be a message space where:

https://t.me/learningnets

$$Exp_{\mathcal{A},\Pi}^{INT-CTX}(1^\lambda)$$

- $K \leftarrow\$ \Pi.\mathsf{Gen}(1^\lambda)$. Initialize $S = \emptyset$, $\mathsf{win} = \mathsf{false}$ and provide oracle access to $\Pi.\mathsf{Enc}_K(\cdot)$ to $\mathcal{A}$.
- For any query M_i, $C_i \leftarrow\$ \Pi.\mathsf{Enc}_K(M_i)$, $S = S \cup \{C_i\}$.
- For any query $VF(C)$, $M \leftarrow\$ \Pi.\mathsf{Dec}_K(C)$. If $M \neq \perp$ and $C \notin S$ return 1 and set $\mathsf{win} = \mathsf{true}$.
- After receiving "Finalize", output win.

Figure 2: INT-CTX for Π

- Gen is the key generation algorithm that takes as input 1^λ and outputs a key pair (pk, sk). Concretely, $(pk, sk) \leftarrow\$ \mathsf{Gen}(1^\lambda)$.
- Enc is the encryption algorithm that takes as input a public key pk and a message m from M and outputs a ciphertext c. That is, $c \leftarrow\$ \mathsf{Enc}(pk, m)$.
- Dec is the decryption algorithm that takes as input a secret key sk and a ciphertext c and outputs a message m or an error $(\perp)$. That is $m = \mathsf{Dec}(sk, c)$.

For completeness, it is required that for every $\lambda \in \mathbb{N}$, every $(pk, sk) \leftarrow\$ \mathsf{Gen}(1^\lambda)$, and every $m \in M$, it holds that $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$ except with negligible probability.

$$\mathsf{PubK}_{\mathcal{A},\Pi_{pub}}^{cpa}(\lambda)$$

1. $(pk, sk) \leftarrow\$ \mathsf{Gen}(1^\lambda)$.
2. $m_0, m_1 \leftarrow \mathcal{A}(pk)$ s.t. $|m_0| = |m_1|$ and $m_0, m_1 \in M$.
3. $b \leftarrow\$ \{0, 1\}$, $c^* \leftarrow\$ \mathsf{Enc}_{pk}(m_b)$.
4. $b' \leftarrow \mathcal{A}(c^*)$.
5. If $b' = b$ output 1, else output 0.

Figure 3: IND-CPA for Π_{pub}

We capture the IND-CPA experiment for public key encryption in Figure 3. We say an encryption scheme Π_{pub} is IND-CPA secure if:

$$\Pr[\mathsf{PubK}_{\mathcal{A},\Pi_{pub}}^{cpa}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

4.3 Digital signature schemes.

Now we revisit the definition from chapter 12 of [KL14]. A digital signature scheme $\Sigma := (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Vf})$ for a message space M is a tuple of three PPT algorithms such that:

- Gen is the key generation algorithm that takes 1^λ as input and outputs a pair of public and private keys (pk, sk). Concretely $(pk, sk) \leftarrow\$ \mathsf{Gen}(1^\lambda)$.
- Sig is the signing algorithm that takes a private key sk and a message m from the message space M as input and outputs a signature σ. We write this as $\sigma \leftarrow\$ \mathsf{Sig}(sk, m)$.
- Vf is the verification algorithm that takes a public key pk, a message m, and a signature σ as input and outputs 1 if a valid signature and 0 if an invalid signature. We write this as $b = \mathsf{Vf}(pk, m, \sigma)$.

For completeness, we require that for $(pk, sk) \leftarrow\$ \mathsf{Gen}(1^\lambda)$, $m \in M$, it holds that $1 = \mathsf{Vf}(pk, m, \mathsf{Sig}(sk, m))$ except with a negligible probability.

We present the game in figure 4 to capture unforgeability. We say a signature scheme Σ is existentially unforgeable against chosen-message attacks (i.e. secure) if for all PPT $\mathcal{A}$ we have that $\Pr[\mathsf{Sig\text{-}Forge}\mathcal{A}, \Sigma(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

https://t.me/learningnets

$\mathsf{Sig\text{-}Forge}\mathcal{A}, \Sigma(\lambda)$

1. $(pk, sk) \leftarrow\$ \mathsf{Gen}(1^\lambda)$.
2. $m, \sigma \leftarrow \mathcal{A}^{\mathsf{Sig}_{sk}(\cdot)}(pk)$ let $\mathcal{Q}$ be the set of queries $\mathcal{A}$ asked to the oracle.
3. If $1 = \mathsf{Vf}(pk, m, \sigma)$ and $m \notin \mathcal{Q}$ output 1; otherwise output 0.

Figure 4: EUF-CMA forΣ

4.4 Commitment Schemes

A commitment scheme $\mathtt{COM}$ is a tuple of algorithms $(\mathsf{Commit}, \mathsf{Open})$ that allows a party to produce a commitment $\mathtt{com}$ to a value x. We make use of a statistically hiding and computationally binding commitment scheme [DF02], defined as:

- A commitment scheme is considered *statistically hiding* if for any x, x' $\mathsf{Commit}(x)$ and $\mathsf{Commit}(x')$ are statistically indistinguishable (as defined by [GMR89]) [DF02]
- A commitment scheme is considered *computationally binding* if the probability that any PPT adversary can produce $\mathtt{com}, \mathtt{open}, \mathtt{open}'$, such that $\mathtt{open} \neq \mathtt{open}'$ but both open $\mathtt{com}$ is less than $\mathsf{negl}(\lambda)$ [KL14]

4.5 Global Clock Functionality

Next, we present the global reference clock functionality $\mathcal{G}_{refClock}$ (Figure 5) [CHMV17]. This functionality is used by our ideal functionality $\mathcal{G}_{Perm}$ to determine the amount of time that has passed between notifying a client of a request for permission on their behalf, and the client responding with an acceptance, denial, or silence.

Functionality: $\mathcal{G}_{refClock}$

Participants: The environment $\mathcal{Z}$, some party P
Variables: An integer G representing the time, initially 0

Procedures:
- **Increment Time**
  - Upon receipt of (`increment time`) from $\mathcal{Z}$, set $G = G + 1$ and send (`incremented`) to $\mathcal{Z}$. Ignore any (`increment time`) from any other party
- **Get Time**
  - Upon receipt of (`get time`) from a party P, send $(\mathtt{time}, G)$ to P

Figure 5: $\mathcal{G}_{refClock}$ The Global Functionality for a Reference Clock

Proof-of-Publication Ledger $\mathcal{L}$. We assume that the parties have access to an unforgeable, verifiable ledger $\mathcal{L}$ as modeled in [KGM19]. The concept of unforgeability here is the same as unforgeability of signatures. Upon posting a transaction tx to a ledger with chain ID cid, the posting party receives an authentication tag σ such that it is hard to compute σ without posting tx to the ledger. cid allows us to identify a specific chain of posts, and for the purpose of our permission protocol we will assume that only posts pertaining to the permission protocol and server S will be made to the chain with ID cid. The structure of cid is dependent on the instantiation of $\mathcal{L}$. At a high level, $\mathcal{L}$ provides the following:

- $(\mathsf{tx}, \sigma) \leftarrow \mathcal{L}.\mathsf{Post}(z, \mathsf{cid})$: This allows a user to post $\mathsf{Data} = z$ on the append only ledger for the chain identifier, cid. We will often use $\mathsf{tx.Data}$ to refer to the contents posted, in this case z. The output of this interface is the transaction, tx, and an authentication tag for verifying that the data was posted on the ledger. The chain identifier ties multiple transactions together.

https://t.me/learningnets

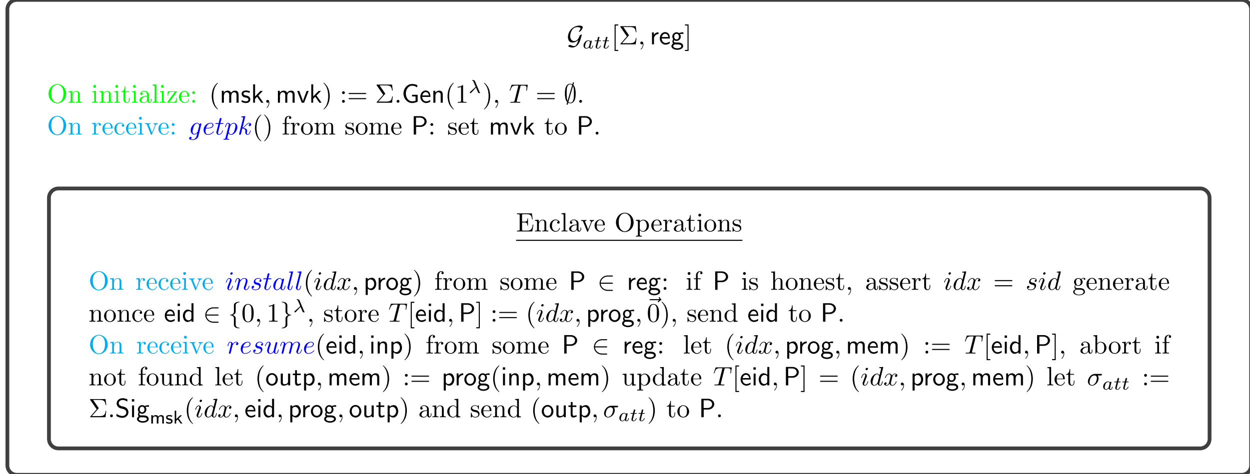

Figure 6: $\mathcal{G}_{att}$ The Ideal Functionality of a TEE

- $\{0,1\} \leftarrow \mathcal{L}.\mathsf{Verify}(\mathsf{tx}, \sigma)$: This allows any user to verify that the pair (tx, σ) were the result of the $\mathcal{L}.\mathsf{Post}$ procedure above.

We use this ledger due to the public verifiability, which allows our TEE to verify transactions *without having access to the ledger*. It is vital that the TEE be able to verify transactions so that a malicious server is not able to produce forged permission.

Trusted Execution Environment. The Cloud has access to a trusted execution environment (TEE) represented as the ideal functionality $\mathcal{G}_{att}$ (Figure 6) presented in [PST17]. Recall that a TEE is an enclave on a computer that can securely perform computation and store data. In Figure 6 all Blue activation points are activation points that can be executed more than once. However, Green activation points can be only executed once. Let prog be the program run by the enclave. Upon performing this computation, the TEE returns an attestation, which is a signature based on keys provided by the manufacturer.

At a high level, $\mathcal{G}_{att}$ is defined for a signature scheme Σ_{att} and registry reg that lists the parties equipped with a processor containing the TEE, these parties are known as the host. Upon initialization (at the factory) the TEE enabled processor is initialized with a signature key pair $\mathsf{msk}, \mathsf{mvk}$. This initialization is only performed once, and the signature pair is used for attesting the execution of a program in the TEE. $\mathcal{G}_{att}$ allows parties to query the public verification key, mvk. In practice, this is often implemented as an online trusted resource where users can verify an attestation from the TEE.

To use the TEE, the host P first calls $install(idx, \mathsf{prog})$ to install the defined program, prog. $\mathcal{G}_{att}$ checks that $\mathsf{P} \in \mathsf{reg}$ and $idx = sid$, where idx is provided by the host and sid is the session ID stored by the TEE. Then it generates a random identifier, eid, to identify the installed enclave and returns eid to party, P.

Next to run the program on any input, inp, the host calls $resume(\mathsf{eid}, \mathsf{inp})$, and the TEE runs the program defined for eid on the input inp. Finally, it returns the output along with a signature under msk as its attestation, which can be verified using the public mvk.

5 Credential-less Publicly Verifiable Permissions

In order to allow for the recovery of a secret without any memory of anything, including a secret key or access to private channels, we need verifiable permissions that are generated in the same setting. Any party can request permission for any registered client, and the permission received will be a certificate for a server to perform some procedure. In secret recovery, this procedure will be recovering a stored secret.

https://t.me/learningnets

Functionality: $\mathcal{G}_{Perm}$

Participants: A set of servers $\mathcal{S}$, a set of clients $\mathcal{C}$, a party P, the adversary $\mathcal{A}$
Variables: L, the set of registered servers, L_S the list of clients registered to server S
External Functionalities: $\mathcal{G}_{refClock}$ the global clock functionality
Algorithms: $\mathsf{VerifyPerm}$ checks the validity of permissions

Procedures:

- **Registration - Server**
  1. Upon receipt of $(\texttt{register server}, \mathsf{S})$ from $\mathsf{S} \in \mathcal{S}$ for the first time, add S to L, set $L_\mathsf{S} = \varnothing$, and send $(\texttt{registered}, \mathsf{S})$ to $\mathcal{C}$ and $\mathcal{A}$
- **Registration - Client**
  1. Upon receipt of $(\texttt{register client}, \mathsf{C}, \mathsf{S})$ from $\mathsf{C} \in \mathcal{C}$ for the first time, send $(\texttt{registration request}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$
  2. Upon receipt of $(\texttt{client perm-info}, \mathsf{C}, \mathsf{perm\text{-}info})$ from $\mathcal{A}$ add $(\mathsf{perm\text{-}info}, \mathsf{S}, \perp)$ to L_S and send $(\mathsf{perm\text{-}info}, \mathsf{S})$ to C, S, and $\mathcal{A}$
- **Generate Permission**
  1. Upon receipt of $(\texttt{generate permission}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req})$ from party P, send $(\texttt{permission request}, \mathsf{perm\text{-}info})$ to S and $\mathcal{A}$
     (a) If $(\mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{res}\|\mathsf{perm}) \notin L_\mathsf{S}$ then output $(\texttt{nonexistent client}, \mathsf{perm\text{-}info})$ to P, S, and $\mathcal{A}$
     (b) Else, if $(\mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{res}\|\mathsf{perm} \neq \perp) \in L_\mathsf{S}$ parse $\mathsf{perm\text{-}info}$ to identify C, then send $(\texttt{existing permissions}, \mathsf{perm\text{-}info})$ to $\mathsf{P}, \mathsf{C}, \mathsf{S}$, and $\mathcal{A}$
     (c) Else send $(\texttt{permission requested}, \mathsf{perm\text{-}info}, \mathsf{S})$ to C and receive a response res, where res can be $\texttt{accepted}, \texttt{denied}$, or $\texttt{silent}$ and let t_{elapse} be the time between sending and receiving a response according to $\mathcal{G}_{refClock}$
        i. If $\mathsf{res} = (\texttt{accepted})$ send $(\texttt{acceptance proof}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}, t_{elapse})$ to $\mathcal{A}$ and receive perm
        ii. Else if $\mathsf{res} = (\texttt{denied})$ send $(\texttt{denial proof}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}, t_{elapse})$ to $\mathcal{A}$ and receive perm
        iii. Else send $(\texttt{silent}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req})$ to $\mathcal{A}$ and receive perm
        iv. Add $(\mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$ to L_S and send $(\texttt{permission}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$ to P and $\mathcal{A}$
- **Verify Permission**
  1. Upon receipt of $(\texttt{verify permission}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}, \mathsf{perm})$ from party P, if there exists $(\mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{res}\|\mathsf{perm}) \in L_\mathsf{S}$
     (a) Let $\mathsf{VerifyPerm}(\mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}, \mathsf{perm}) = b_{ver}$
     (b) If $\mathsf{res} = \texttt{accepted}$ output $(\texttt{accepted}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{perm})$ to P and $\mathcal{A}$
     (c) Else if $\mathsf{res} = \texttt{denied}$ output $(\texttt{denied}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{perm})$ to P and $\mathcal{A}$
     (d) Else if $\mathsf{res} = \texttt{silence}$, if $b_{ver} = 1$ output $(\texttt{accepted}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{perm})$ to P and $\mathcal{A}$. Else output $(\texttt{denied}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{perm})$ to P and $\mathcal{A}$
  2. Else output $(\texttt{denied}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{perm})$ to P and $\mathcal{A}$

Figure 7: $\mathcal{G}_{Perm}$ The Global Functionality for Verifiable Permission

5.1 Definition of Credential-less Publicly Verifiable Permissions

In Figure 7 we present $\mathcal{G}_{Perm}$ the ideal functionality for authenticated credential-less permission generation. In this functionality a client C registers to a server S with public information $\mathsf{perm\text{-}info}$. This information is not secret and should be accessible to the client even after losing all secrets. In this functionality, any party can request publicly-verifiable, unforgeable permissions on behalf of any registered client. This client has the opportunity to accept or deny the permission, or to request that they be generated by silence, showing that

https://t.me/learningnets

Protocol: Π_{Perm} - Register and Manage Permissions

- **Register - Server**
  - S: Upon receiving $(\texttt{register server}, \mathsf{S})$ from $\mathcal{Z}$
    1. Set $\mathbb{T}_{\mathsf{S}} = \varnothing$
    2. Generate signature keys: $(\mathsf{vk_S}, \mathsf{sk_S}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$ and choose a chain id cid.
    3. Post verification key on the ledger: $\mathsf{tx_S}, \sigma_{\mathsf{S}} \leftarrow \mathcal{L}.\mathsf{Post}(\text{"}register\text{"} \| \mathsf{vk_S}, \mathsf{cid})$
- **Register - Client**
  - C: Upon receiving $(\texttt{register client}, \mathsf{S})$ from $\mathcal{Z}$, find $\mathsf{tx_S}, \sigma_{\mathsf{S}}$ on $\mathcal{L}$ with $\mathsf{tx.Data} = \text{"}register\text{"} \| \mathsf{vk_S}$
    1. Generate signature keys: Set $(\mathsf{vk_C}, \mathsf{sk_C}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
    2. Choose t_{open}, t_{chal} as the time allotted to open a commitment and the time allotted to challenge a permission respectively
    3. Set permission identity: $\mathsf{perm\text{-}info} = (\mathsf{C}, \mathsf{vk_C}, t_{open}, t_{chal}, \mathsf{vk_S})$
    4. Post registration on the ledger: $\mathsf{tx_C}, \sigma_{\mathsf{C}} \leftarrow \mathcal{L}.\mathsf{Post}(\mathsf{perm\text{-}info}, \mathsf{cid})$ and send $\mathsf{tx_C}, \sigma_{\mathsf{C}}$ to S
  - S : Upon receiving $(\mathsf{tx_C}, \sigma_{\mathsf{C}})$ from C
    1. Check for prior registration: If $\mathsf{perm\text{-}info} \in L_{\mathsf{S}}$ abort
    2. Else add $\mathsf{perm\text{-}info}$ to L_{S}
    3. Authorize registration: Compute $\sigma_{sig} = \Sigma.\mathsf{Sign}(\mathsf{sk_S}, \mathsf{tx_C.Data})$
    4. Post signature: $(\mathsf{tx}_{reg}, \sigma_{reg}) = \mathcal{L}.\mathsf{Post}(\mathsf{tx_C.Data} \| \sigma_{sig}, \mathsf{cid})$ and send $(\mathsf{tx}_{reg}, \sigma_{reg})$ to C
  - C : Upon receipt of $(\mathsf{tx}_{reg}, \sigma_{reg})$ from S
    1. If $\Sigma.\mathsf{Vf}(\mathsf{vk_S}, \sigma_{sig}, \mathsf{tx_C.Data}) \neq 1$ abort
- **Manage Permissions:** (Run continuously by C upon registration)
  - C : Upon seeing any transaction tx referencing C posted to the ledger
    1. If $\mathcal{Z}$ sends $(\texttt{accepted}, \mathsf{tx})$
       (a) Sign the transaction to accept: $\sigma_{acc-tx} = \Sigma.\mathsf{Sign}(\mathsf{sk_C}, \text{"}accepted\text{"} \| \mathsf{tx.Data})$
       (b) Post acceptance: $(\mathsf{tx}_{acc}, \sigma_{acc}) = \mathcal{L}.\mathsf{Post}(\text{"}accepted\text{"} \| \mathsf{tx.Data} \| \sigma_{acc-tx} \| \mathsf{tx_C.Data})$
    2. Else if $\mathcal{Z}$ sends $(\texttt{denied}, \mathsf{tx})$
       (a) Sign the transaction to deny: $\sigma_{den-tx} = \Sigma.\mathsf{Sign}(\mathsf{sk_C}, \text{"}denied\text{"} \| \mathsf{tx.Data})$
       (b) Post denied: $(\mathsf{tx}_{den}, \sigma_{den}) = \mathcal{L}.\mathsf{Post}(\text{"}denied\text{"} \| \mathsf{tx.Data} \| \sigma_{den-tx} \| \mathsf{tx_C.Data})$
    3. Else do nothing

Figure 8: Registration and Manage Permissions Procedures of Verifiable Permission Protocol Π_{Perm}

the permissions are valid because the client did not deny. With public-verifiability we protect a server from accusations of cheating by a malicious client, who claims that the server performed the procedure without valid permission.

5.2 Realizing $\mathcal{G}_{Perm}$

Here we present our protocol Π_{Perm} to realize $\mathcal{G}_{Perm}$ our ideal functionality for verifiable, credential-less permissions. We realize this functionality in Figures 8 and 9 using a verifiable ledger $\mathcal{L}$, as described by Kaptchuk, Miers, and Green [KGM19]. We consider a setting in which we have many clients $\mathcal{C}$ where each $\mathsf{C} \in \mathcal{C}$ contracts some server $\mathsf{S} \in \mathcal{S}$, such that S needs permission to perform some task, which we represent as a request req. However, because the permission requires no secret, they can be requested by any party.

Registration and Managing Permissions. To begin the parties must register (Figure 8). A server must register by announcing to the world that they are open for registration by posting their verification key and to the ledger with chain ID cid[7]. Once the server has made this announcement, a client may register

[7]We assume that cid is publicly known and associated with this server such that it is easy for a client to find

https://t.me/learningnets

with them by constructing their permission information perm-info. This information contains the client's verification key, the time allotted for opening a commitment t_{open} and challenging a request t_{chal}, and the verification key of the server. Once the client posts this information to the ledger, if they are not already registered, the server accepts the registration by signing the client registration transaction and posting the signature to the ledger (within time t_{chal}). Upon completion of registration, the client will begin to monitor the ledger (Manage Permissions, Figure 8).

Generate Permission. To generate permission (Figure 9) any party may post a commitment transaction and subsequent opening to the request. Once the opening is posted, the client may accept, deny, or remain silent. The server, upon seeing the opening, constructs the permission.

The permission includes the registration transaction of the client and the t_{chal} blocks after (denoted $\mathsf{chalwindow}_{\mathsf{C}}$). Permission also includes a "commitment window", which includes the t_{open} blocks before and ζ blocks after the opening transaction. This set of blocks (denoted $\mathsf{comwindow}_{\mathsf{req}}$) must include the commitment transaction that has been opened. Finally, the permission includes a "challenge window", consisting of the t_{chal} blocks after the opening transaction (denoted $\mathsf{chalwindow}_{\mathsf{req}}$). If there is a signed denial in $\mathsf{chalwindow}_{\mathsf{C}}$, the registration is considered invalid and the request is denied. $\mathsf{chalwindow}_{\mathsf{req}}$ may contain either a signed denial or signed acceptance of a request, and the request will be denied or accepted respectively. $\mathsf{chalwindow}_{\mathsf{req}}$ might also contain an opening for the same request. If this is the case, and there is a valid commitment posted in $\mathsf{comwindow}_{\mathsf{req}}$ for this second opening, we consider these to be competing requests and neither is accepted. If none of these are true, then $\mathsf{comwindow}_{\mathsf{req}}$ is checked to ensure that the commitment to the original request was posted and subsequently opened within the appropriate time frame. In the case of valid permission, the server signs the permission and posts the signature to the ledger.

Verify Permission. Any party must be able to verify permissions, even without access to the ledger. To verify (Figure 10) any party takes the permission and checks that the server and client registrations are valid, that all transactions in the permission verify and are in the correct sequence, and if there exists an acceptance or denial transaction in $\mathsf{chalwindow}_{\mathsf{req}}$.

Other Realizations of $\mathcal{G}_{Perm}$. In Figures 8, 9 we provide our candidate realization, and here we provide a high level overview of other potential realizations. The problem of realizing $\mathcal{G}_{Perm}$ can be narrowed down to the problem of proving that one has the right to request permission for a certain request. If the client still holds some credentials they could use these credentials to prove their identity and therefore their right to the permissions [MKE22]. If the client remembers nothign, they could instead designate a set of parties to serve as vouchers for them [Sch]. These parties, who presumably still hold their own authentication information, would then sign off on a request to verify that it came from the correct source. Another approach is to use a trusted party who has a direct line of communication with the user. If the client needs permission and has lost access to everything, including the private channel, the private channel is used to generate a proof of silence.

Competing Requests. Commit and reveal helps mitigate front-running. However, an adversary may post frequent requests in the hope that one slips through, or may learn in real time that the client has lost access to their credentials. In the case where two competing requests are made, we opt to accept neither. This effectively allows the adversary to block a client's request. We give two potential approaches to mitigating this problem below.

Request Fees. One approach is to institute fees for all requests, as is done in KELP [BCC+21]. In this approach, parties making a request pay an additional fee, on top of the gas fee for posting to the ledger, for both their commitment and opening transactions. This fee does not go to miners[8], but instead goes to the wallet associated with C. This would help deter an adversary from repeatedly posting commitments in the hopes that C eventually makes their own request, as the adversary would lose money in the process, and would potentially never successfully block a request. This provides C with the ability to block the adversary's request as well, *even without knowledge of any secret.*

[8]In the case that a commitment is not opened, this extra does go to the miners

https://t.me/learningnets

Protocol: Π_{Perm} - Generate and Verify Permission

- **Generate Permission**
  - A party P: Upon receiving ($\texttt{generate permissions}, \mathsf{perm\text{-}info}, \mathsf{S}, \texttt{req}$) from $\mathcal{Z}$
    1. Compute a commitment to the request: $(\texttt{com}, \texttt{open}) = \mathsf{Commit}(\mathsf{perm\text{-}info} \| \mathsf{req} \| \mathsf{S} \| \mathsf{tx_C})$
    2. Post commitment to the ledger: $\mathsf{tx}_{com}, \sigma_{com} \leftarrow \mathcal{L}.\mathsf{Post}(\texttt{com}, \mathsf{cid})$
    3. Open the commitment and post the opening: Upon seeing tx_{com} posted to the ledger, post $\mathsf{tx}_{open}, \sigma_{open} \leftarrow \mathcal{L}.\mathsf{Post}(\texttt{open}, \mathsf{cid})$
  - C: Upon seeing tx_{open} on $\mathcal{L}$ where $\mathsf{perm\text{-}info} \in \mathsf{tx}_{open}.\mathsf{Data}$
    1. If $\mathcal{Z}$ sends ($\texttt{accepted}$)
       (a) Sign the opening to accept the request: $\sigma_{granted} = \Sigma.\mathsf{Sign}(\mathsf{sk_C}, \text{"}\textit{accepted}\text{"} \| \mathsf{tx}_{open}.\mathsf{Data})$
       (b) Post the acceptance: $\mathsf{tx}_{accept}, \sigma_{accept} = \mathcal{L}.\mathsf{Post}(\text{"}\textit{accepted}\text{"} \| \mathsf{tx}_{open}.\mathsf{Data} \| \sigma_{granted}, \mathsf{cid})$
    2. Else if $\mathcal{Z}$ sends ($\texttt{denied}$)
       (a) Sign the opening to deny the request: $\sigma_{refuse} = \Sigma.\mathsf{Sign}(\mathsf{sk_C}, \text{"}\textit{denied}\text{"} \| \mathsf{tx}_{open}.\mathsf{Data})$
       (b) Post the denial: $\mathsf{tx}_{denied}, \sigma_{denied} = \mathcal{L}.\mathsf{Post}(\text{"}\textit{denied}\text{"} \| \mathsf{tx}_{open}.\mathsf{Data} \| \sigma_{refuse}, \mathsf{cid})$
    3. Else if $\mathcal{Z}$ sends ($\texttt{silence}$), do nothing
  - S: Upon seeing tx_{open} on $\mathcal{L}$ where $\mathsf{perm\text{-}info} \in \mathsf{tx}_{open}.\mathsf{Data}$ and $\mathsf{perm\text{-}info} \in L_{\mathsf{S}}$
    1. Set ledger blocks as challenge windows:
       (a) Let $\mathsf{chalwindow_C}$ be the t_{chal} blocks after $\mathsf{tx_C}$
       (b) Let $\mathsf{comwindow_{req}}$ be the t_{open} blocks before and the ζ blocks after tx_{open}, and $\mathsf{chalwindow_{req}}$ be the t_{chal} blocks after tx_{open} including tx_{open}
    2. Verify registration transaction:
       (a) Check for acceptance: If there exists $\mathsf{tx}_{acc} \in \mathsf{chalwindow_C}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk_C}, \sigma_{acc-tx}, \text{"}\textit{accepted}\text{"} \| \mathsf{tx_C}) = 1$ for $\sigma_{acc-tx} \in \mathsf{tx_C}.\mathsf{Data}$, set $\mathsf{chalwindow_C}$ to be the ledger blocks from $\mathsf{tx_C}$ to tx_{acc}
       (b) Check for denial: If there exists $\mathsf{tx}_{den} \in \mathsf{chalwindow_C}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk_C}, \sigma_{den-tx}, \text{"}\textit{denied}\text{"} \| \mathsf{tx_C}) = 1$ for $\sigma_{den-tx} \in \mathsf{tx_C}.\mathsf{Data}$, set $\mathsf{chalwindow_C}$ to be the ledger blocks from $\mathsf{tx_C}$ to tx_{den}
    3. Check for valid commitment: If there does not exist $\mathsf{tx}_{com} \in \mathsf{comwindow_{req}}$ such that $\mathsf{tx}_{open}.\mathsf{Data}$ is a valid opening of $\mathsf{tx}_{com}.\mathsf{Data}$, output $\perp$ and abort
    4. Verify permission request:
       (a) Check for acceptance: If there exists $\mathsf{tx}_{accept} \in \mathsf{chalwindow_{req}}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk_C}, \sigma_{granted}, \text{"}\textit{accepted}\text{"} \| \mathsf{tx}_{open}.\mathsf{Data}) = 1$ for $\sigma_{granted} \in \mathsf{tx}_{accept}.\mathsf{Data}$, set $\mathsf{chalwindow_{req}}$ to be the ledger blocks from tx_{open} to tx_{accept}
       (b) Check for denial: Else if there exists $\mathsf{tx}_{denied} \in \mathsf{chalwindow_{req}}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk_C}, \sigma_{refuse}, \text{"}\textit{denied}\text{"} \| \mathsf{tx}_{open}.\mathsf{Data}) = 1$ for $\sigma_{refuse} \in \mathsf{tx}_{denied}.\mathsf{Data}$, set $\mathsf{chalwindow_{req}}$ to be the ledger blocks from tx_{open} to tx_{denied}
       (c) Check for competing requests: Else if there exists $\mathsf{tx}'_{open} \in \mathsf{chalwindow_{req}}$ such that there exists $\mathsf{tx}'_{com} \in \mathsf{comwindow_{req}}$ where $\mathsf{tx}'_{open}.\mathsf{Data}$ is a valid opening of $\mathsf{tx}'_{com}.\mathsf{Data}$ for the same $\mathsf{perm\text{-}info}$ or a valid opening of tx_{com}, the distance between tx'_{open} and the commitment it opens is less than or equal to t_{open}, output $\perp$ and abort
    5. Sign the permissions: Compute $\sigma_{perm} = \Sigma.\mathsf{Sign}(\mathsf{sk_S}, \mathsf{tx_C} \| \sigma_{\mathsf{C}} \| \mathsf{chalwindow_C} \| \texttt{open} \| \mathsf{comwindow_{req}} \| \mathsf{chalwindow_{req}})$ set $\mathsf{perm} = \mathsf{tx_S} \| \sigma_{\mathsf{S}} \| \mathsf{tx_C} \| \sigma_{\mathsf{C}} \| \mathsf{chalwindow_C} \| \texttt{open} \| \mathsf{comwindow_{req}} \| \mathsf{chalwindow}_{req} \| \sigma_{perm}$ and post $\mathsf{tx}_{fin}, \sigma_{fin} = \mathcal{L}.\mathsf{Post}(\mathsf{perm}, \mathsf{cid})$
- **Verify Permission**
  - A party P : Upon receipt of ($\texttt{verify permission}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}, \mathsf{perm}$) from $\mathcal{Z}$
    1. Run $\mathsf{VerifyPerm}(\mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}, \mathsf{perm})$

Figure 9: Generate and Verify Permission Procedures of Verifiable Permission Protocol Π_{Perm}

https://t.me/learningnets

Algorithm: $\{0,1\} \leftarrow \mathsf{VerifyPerm}(\mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}, \mathsf{perm})$

1. Parse $\mathsf{perm} = \mathsf{tx_S} \| \sigma_\mathsf{S} \| \mathsf{tx_C} \| \sigma_\mathsf{C} \| \mathsf{chalwindow_C} \| \mathsf{open} \| \mathsf{comwindow_{req}} \| \mathsf{chalwindow_{req}} \| \sigma_{perm}$
2. Parse $\mathsf{tx_S}.\mathsf{Data} = \text{“}register\text{”} \| \mathsf{vk_S}$ and $\mathsf{tx_C}.\mathsf{Data} = \mathsf{perm\text{-}info}'$
3. If $\mathsf{perm\text{-}info}' \neq \mathsf{perm\text{-}info} = (\mathsf{C}, \mathsf{vk_C}, t_{open}, t_{chal}, \mathsf{vk_S})$ output 0
4. Check server signature: If $\Sigma.\mathsf{Vf}(\mathsf{vk_S}, \sigma_{perm}, \mathsf{tx_C} \| \sigma_\mathsf{C} \| \mathsf{chalwindow_C} \| \mathsf{open} \| \mathsf{comwindow_{req}} \| \mathsf{chalwindow_{req}}) \neq 1$ output 0
5. Check $\mathsf{perm\text{-}info}$ of permission: If $\mathsf{perm\text{-}info}$ in $\mathsf{tx_C}$ is not the same as the $\mathsf{perm\text{-}info}$ in open, output 0
6. Check request: If $\mathsf{req} \notin \mathsf{open}$, output 0
7. Verify registration transactions: If $\mathcal{L}.\mathsf{Vf}(\mathsf{tx_S}, \sigma_\mathsf{S}) \neq 1$ or $\mathcal{L}.\mathsf{Vf}(\mathsf{tx_C}, \sigma_\mathsf{C}) \neq 1$ output 0
8. Verify transactions from commitment window and challenge window: If there exists any pair (tx, σ) in $\mathsf{chalwindow_C}$ $\mathsf{comwindow_{req}}$ or $\mathsf{chalwindow_{req}}$ such that $\mathcal{L}.\mathsf{Vf}(\mathsf{tx}, \sigma) \neq 1$ output 0
9. Verify the order of the transactions: If the transactions in $\mathsf{chalwindow_C}$, $\mathsf{comwindow_{req}}$ or $\mathsf{chalwindow_{req}}$ are not a direct sequence, output 0
10. Check client registration:
    (a) If $\mathsf{tx_C} \in \mathsf{perm}$ is not in open, output 0
    (b) If there does not exist $\mathsf{tx}_{reg} \in \mathsf{chalwindow_C}$ with $\sigma_{sig} \in \mathsf{tx}_{reg}.\mathsf{Data}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk_S}, \sigma_{sig}, \mathsf{tx_C}.\mathsf{Data}) = 1$ output 0
    (c) Check for denial: If there exists $\mathsf{tx}_{den} \in \mathsf{chalwindow_C}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk_C}, \sigma_{den-tx}, \text{“}denied\text{”} \| \mathsf{tx_C}) = 1$ for $\sigma_{den-tx} \in \mathsf{tx_C}.\mathsf{Data}$ output 0
11. Check opening and commitment: If there does not exists $\mathsf{tx}_{open} \in \mathsf{chalwindow_{req}}$ such that $\mathsf{tx}_{open}.\mathsf{Data} = \mathsf{open}$ is a valid opening of $\mathsf{tx}_{com} \in \mathsf{comwindow_{req}}$, output 0
12. Else
    (a) If there exists $\mathsf{tx}_{denied} \in \mathsf{chalwindow_{req}}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk_{Client}}, \sigma_{refuse}, \text{“}denied\text{”} \| \mathsf{tx}_{open}.\mathsf{Data}) = 1$ for $\sigma_{refuse} \in \mathsf{tx}_{denied}.\mathsf{Data}$ output 0
    (b) If there exists $\mathsf{tx}_{accept} \in \mathsf{chalwindow_{req}}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk_{Client}}, \sigma_{granted}, \text{“}accepted\text{”} \| \mathsf{tx}_{open}.\mathsf{Data}) = 1$ for $\sigma_{granted} \in \mathsf{tx}_{accept}.\mathsf{Data}$ output 1
    (c) Else if there exists $\mathsf{tx}'_{open} \in \mathsf{chalwindow_{req}}$ such that there exists $\mathsf{tx}'_{com} \in \mathsf{comwindow_{req}}$ where $\mathsf{tx}'_{open}.\mathsf{Data}$ is a valid opening of $\mathsf{tx}'_{com}.\mathsf{Data}$ for the same $\mathsf{perm\text{-}info}$ or a valid opening of tx_{com}, the distance between tx'_{open} and the commitment it opens is less than or equal to t_{open}, output 0
13. Else if $|\mathsf{comwindow_{req}}| = t_{open} + \zeta$ and $|\mathsf{chalwindow_{req}}| = t_{chal}$, output 1

Figure 10: The Algorithm VerifyPerm for Local Verification of Permission

https://t.me/learningnets

Second Line of Defense. Another approach is to institute a second line of defense, where upon competing recoveries, the two parties enter an authentication game as demonstrated by Maram et al. [MKE22]. Maram et al. describe multiple ways to instantiate such a game, where C competes against the adversary by showing knowledge credentials or other facts about themselves that they should know.

5.3 Instantiating Π_{Perm}

In order to implement Π_{Perm}, we need a SUF-AUTH secure ledger, a statistically hiding and computationally binding commitment scheme, and a EUF-CMA secure signature scheme.

A detailed discussion on instantiating this SUF-AUTH ledger was provided by Kaptchuk et al. [KGM19], we give a high level overview here but refer the reader to their work for details.

The ledger is required to store a state and provide a method of verifying parts of this state. This verification must show that certain information is part of the state and allow one to confirm that a sequence of blocks appear in a particular order. A straightforward realization of the ledger is a permissioned blockchain. In a permissioned blockchain, only certain miners are permitted to contribute blocks, which are verified using the miners' verification keys, and blockchains inherently contain information to confirm the sequence of blocks. In this realization, assuming that the ledger is only used for a single purpose, the cid of the ledger consists of the verification keys of the miners who are permitted to add blocks to the ledger. The authentication tag σ is a signature on the contents of the post tx under at least one of these verification keys, and includes cid, allowing for easy verification.

Verifiable ledgers can also be realized using a permissionless blockchain. A Bitcoin-like blockchain can be used to build a verifiable ledger by including what Kaptchuk et al. call confirmation blocks, which are blocks immediately following a post to ensure that a transaction makes it to the chain. The ledger can also be realized using an smart contract system like Ethereum, where the state is maintained and updated by a smart contract.

While not required in the original work, we only consider realizations that are *computationally* unforgeable. Thus solutions such as permissionless blockchains are not suitable for our setting, as the chains are forgeable, although at a steep economic cost. We consider only computationally unforgeable ledgers due to the consequences of a forged block in our setting. Specifically, if a client were to store the key to a cryptocurrency wallet that contains a significant amount of money, the economic cost of computing forgeries may not be enough of a deterrent to stop an adversary from forging blocks to create phony permission.

For a permissioned ledger, we consider the example of Ripple [Rip]. In Ripple, a signed transaction (with four signatures) costs at a minimum 50 "drops"[9]. A drop is equivalent to .00001 XRP where XRP is the on chain currency. At the time of writing this paper, XRP is worth 0.7 USD. Therefore, the cost of a single post is extremely low.

A client must post at least a registration transaction. The client may need to decline malicious requests for permission. However, an implementation of a request fee paid to the rightful client upon denial would mean that the client makes money on these transactions rather than lose money. The client will post a commit and reveal transaction upon requesting permission. Assuming the cost of 50 drops, this will cost a total of .0015 XRP or 0.00105 USD. No computation is performed on the ledger, therefore we need only worry about the costs of posting.

The server must also post to the ledger. Once upon registration, and once upon successful generation of permission. While the cost of this will be extremely low, a server may charge a fee to any client registered to them to cover this cost, as well as the cost of computation.

Pedersen commitments [Ped91] provide all that we need from our commitment scheme. The client will need to post a minimum of *one commitment* upon the need to generate permission. Lastly we need a EUF-CMA secure encryption scheme, such as the ElGamal signature scheme [Gam84], with a slight modification in the random oracle model [Bon11]. The client may need to sign denials on invalid requests, request fees will mean the client gets paid for this work. The client may need to sign *once* to accept permission. They will also need to sign to verify the generation of permission.

[9] No maximum cost is given. The cost of a signed transaction is dependent on the number of signatures, with 10 drops being added per signature.

https://t.me/learningnets

Simulator: $\mathsf{Sim}_{\mathsf{S}^*}$ - Registration

- **Register - Server:** Upon query ("*register*"$\|\mathsf{vk}_\mathsf{S}$) to $\mathsf{OLedger}$
  1. Send ($\texttt{register server}, \mathsf{S}^*$) to $\mathcal{G}_{Perm}$
  2. Forward the query to $\mathsf{OLedger}$ and receive $\mathsf{tx}_{\mathsf{S}^*}, \sigma_{\mathsf{S}^*}$, store this in L_{cid} and forward to S^*
- **Register - Client:** Upon receipt of ($\texttt{registration request}, \mathsf{C}, \mathsf{S}^*$) from $\mathcal{G}_{Perm}$
  1. Set $(\mathsf{vk}_{\mathsf{sim}}, \mathsf{sk}_{\mathsf{sim}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
  2. Choose t_{open}, t_{chal} as the time to open a commitment and challenge a request respectively
  3. Set $\mathsf{perm\text{-}info} = (\mathsf{C}, \mathsf{vk}_{\mathsf{sim}}, t_{open}, t_{chal}, \mathsf{vk}_\mathsf{S})$ and send ($\texttt{client perm-info}, \mathsf{C}, \mathsf{perm\text{-}info}$)
  4. Query $\mathsf{OLedger}$ with ($\mathsf{perm\text{-}info}$), receive $\mathsf{tx}_\mathsf{C}, \sigma_\mathsf{C}$, store in L_{cid}, and send $\mathsf{tx}_\mathsf{C}, \sigma_\mathsf{C}$ to S^*
  5. Upon query $(\mathsf{tx}_\mathsf{C}.\mathsf{Data}\|\sigma_{sig})$ to $\mathsf{OLedger}$ by S^*, if $\Sigma.\mathsf{Vf}(\mathsf{vk}_\mathsf{S}, \sigma_{sig}, \mathsf{tx}_\mathsf{C}.\mathsf{Data}) \neq 1$ abort, else forward the query to receive $(\mathsf{tx}_{reg}, \sigma_{reg})$ and store in L_{cid}

Figure 11: Simulation of the Register Procedure of Π_{Perm} for a Malicious Server S^*

5.4 Security Proofs

Here we present our Theorem 1 on the security of the protocol Π_{Perm} and provide a sketch of the proof.

Theorem 1. *If $\mathcal{L}$ is a* SUF-AUTH *ledger represented as an oracle* OLedger*,* $\texttt{COM} = (\mathsf{Commit}, \mathsf{Open})$ *is a statistically hiding, computationally binding commitment scheme, and* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$ *is a* EUF-CMA *signature scheme, then Π_{Perm} realizes the ideal functionality $\mathcal{G}_{Perm}$*

Case: Malicious Server First we consider the case of a malicious server. To prove security in this case, we present the simulator $\mathsf{Sim}_{\mathsf{S}^*}$ (Figures 11, 12, 13) that generates the view for a malicious S^* in the ideal world. We then prove, through a series of hybrids, that the view generated by this simulator is indistinguishable from the view of a malicious server in the real world executing Π_{Perm}.

Proof by Hybrids

We prove that the view simulated by $\mathsf{Sim}_{\mathsf{S}^*}$ is indistinguishable from a view of the adversary in the real world through a series of hybrids, starting from the real world protocol and moving step-by-step until we reach the ideal world. By proving that each hybrid is indistinguishable from the last, we will prove that the real and ideal world are indistinguishable to a malicious S^*.

- $\mathbf{Hyb}_0$: The real world protocol
- $\mathbf{Hyb}_1$: This is the same as $\mathbf{Hyb}_0$, except that $\mathsf{Sim}_{\mathsf{S}^*}$ aborts with **ForgeFail** when S^* submits forget blocks as permissions
- $\mathbf{Hyb}_2$: This is the same as $\mathbf{Hyb}_1$ except that $\mathsf{Sim}_{\mathsf{S}^*}$ aborts with **SigForge** when S^* submits a signature on behalf of an honest client
- $\mathbf{Hyb}_3$: This is the same as $\mathbf{Hyb}_2$ except that $\mathsf{Sim}_{\mathsf{S}^*}$ aborts with **CommitFail** when there are two valid requests for the same $\mathsf{perm\text{-}info}$ in the permissions

Lemma 1. *If* OLedger *is realized as a* SUF-AUTH *secure ledger $\mathcal{L}$,* $\mathbf{Hyb}_0$ *is indistinguishable from* $\mathbf{Hyb}_1$

Proof. Note that the concept of unforgeability here is the same as the concept of unforgeability of signatures. That is, the adversary wins if they are able to produce a pair $(\mathsf{tx}^*, \sigma^*)$ that verifies, but was not the result of a call $\mathcal{L}.\mathsf{Post}$.

Towards a contradiction, assume that there exists a PPT adversary $\mathcal{A}$ such that $|Pr[\mathcal{A}(\mathbf{Hyb}_1) = 1] - Pr[\mathcal{A}(\mathbf{Hyb}_0) = 1]| > \mathsf{negl}(\lambda)$. The only difference between these two hybrids is that in $\mathbf{Hyb}_1$, $\mathsf{Sim}_{\mathsf{S}^*}$ aborts if S^* submits permissions that include blocks that were not the result of a query to $\mathsf{OLedger}$. The only way $\mathcal{A}$ could distinguish between these two hybrids is if they can produce blocks $(\mathsf{tx}^*, \sigma^*)$ that verify but were not posted on the ledger.

Therefore, we can use $\mathcal{A}$ to construct a reduction $\mathcal{D}$ such that $\mathcal{D}$ can win the unforgeability game for a SUF-AUTH ledger. Define $\mathcal{D}$ as follows:

$\underline{\mathcal{D}(\mathsf{cid})}$:

https://t.me/learningnets

Simulator: $\mathsf{Sim}_{\mathsf{S}^*}$ - Generate Permission Honest Request

- **Generate Permission**: Upon receipt of ($\texttt{permission request}$, perm-info) from $\mathcal{G}_{Perm}$
  1. Upon receipt of $(\mathsf{res}, \mathsf{perm\text{-}info}, \mathsf{S}^*, \mathsf{req}, t_{elapse})$ from $\mathcal{G}_{Perm}$
  2. Query OLedger with $\texttt{com}$, where $(\texttt{com}, \texttt{open}) = \mathsf{Commit}(\mathsf{perm\text{-}info}\|\mathsf{req}\|\mathsf{S}^*\|\mathsf{tx_C})$, receive $\mathsf{tx}_{com}, \sigma_{com}$ and store in L_{cid}
  3. After time t_{elapse}, query OLedger with $\texttt{open}$, and store $\mathsf{tx}_{open}, \sigma_{open}$ in L_{cid}
  4. If $\mathsf{res} = \texttt{acceptance proof}$
     (a) Compute $\sigma_{granted} = \Sigma.\mathsf{Sign}(\mathsf{sk_{sim}}, \text{“}accepted\text{”}\|\mathsf{tx}_{open}.\mathsf{Data})$, query OLedger with “*accepted*”$\|\mathsf{tx}_{open}.\mathsf{Data}\|\sigma_{granted}$, receive $\mathsf{tx}_{accept}, \sigma_{accept}$ and store in L_{cid}
     (b) Upon query $\mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx_C}\|\sigma_\mathsf{C}\|\mathsf{chalwindow_C}\|\mathsf{open}\|\mathsf{comwindow_{req}}\|\mathsf{chalwindow_{req}}\|\mathsf{vk_S}\|\sigma_{perm}$ to OLedger by S^*, if $\Sigma.\mathsf{Vf}(\mathsf{vk_S}, \sigma_{perm}, \mathsf{tx_C}\|\sigma_\mathsf{C}\|\mathsf{chalwindow_C}\|\mathsf{open}\|\mathsf{comwindow_{req}}\|\mathsf{chalwindow_{req}})$ $\neq 1$ abort, else
        i. If $(\mathsf{tx}_{accept}, \sigma_{accept}) \notin \mathsf{chalwindow}$ abort with **ForgeFail**
        ii. If there exists $(\mathsf{tx}_{denied}, \sigma_{denied}) \in \mathsf{chalwindow}$ with $\sigma_{refuse} \in \mathsf{tx}_{denied}.\mathsf{Data}$ and $\Sigma.\mathsf{Vf}(\mathsf{vk_{sim}}, \sigma_{refuse}, \text{“}denied\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$ abort with **SigForge**
  5. Else if $\mathsf{res} = \texttt{denial proof}$
     (a) Compute $\sigma_{refuse} = \Sigma.\mathsf{Sign}(\mathsf{sk_{sim}}, \text{“}denied\text{”}\|\mathsf{tx}_{open}.\mathsf{Data})$, query OLedger with “*denied*”$\|\mathsf{tx}_{open}.\mathsf{Data}\|\sigma_{refuse}$, receive $\mathsf{tx}_{denied}, \sigma_{denied}$ and store in L_{cid}
     (b) Upon query $\mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx_C}\|\sigma_\mathsf{C}\|\mathsf{chalwindow_C}\|\mathsf{open}\|\mathsf{comwindow_{req}}\|\mathsf{chalwindow_{req}}\|\mathsf{vk_S}\|\sigma_{perm}$ to OLedger by S^*, if $\Sigma.\mathsf{Vf}(\mathsf{vk_S}, \sigma_{perm}, \mathsf{tx_C}\|\sigma_\mathsf{C}\|\mathsf{chalwindow_C}\|\mathsf{open}\|\mathsf{comwindow_{req}}\|\mathsf{chalwindow_{req}})$ $\neq 1$ abort, else
        i. If $(\mathsf{tx}_{denied}, \sigma_{denied}) \notin \mathsf{chalwindow}$ abort with **ForgeFail**
        ii. If there exists $(\mathsf{tx}_{accept}, \sigma_{accept}) \in \mathsf{chalwindow}$ with $\sigma_{granted} \in \mathsf{tx}_{accept}.\mathsf{Data}$ and $\Sigma.\mathsf{Vf}(\mathsf{vk_{sim}}, \sigma_{granted}, \text{“}accepted\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$ abort with **SigForge**
  6. Else
     (a) Upon query $\mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx_C}\|\sigma_\mathsf{C}\|\mathsf{chalwindow_C}\|\mathsf{open}\|\mathsf{comwindow_{req}}\|\mathsf{chalwindow_{req}}\|\mathsf{vk_S}\|\sigma_{perm}$ to OLedger by S^*, if $|\mathsf{comwindow}| \neq t_{open} + \zeta$, $|\mathsf{chalwindow}| \neq t_{chal}$, or $\Sigma.\mathsf{Vf}(\mathsf{vk_S}, \sigma_{perm}, \mathsf{tx_C}\|\sigma_\mathsf{C}\|\mathsf{chalwindow_C}\|\mathsf{open}\|\mathsf{comwindow_{req}}\|\mathsf{chalwindow_{req}}) \neq 1$ abort
        i. If there exists $(\mathsf{tx}_{accept}, \sigma_{accept}) \in \mathsf{chalwindow}$ with $\sigma_{granted} \in \mathsf{tx}_{accept}.\mathsf{Data}$ and $\Sigma.\mathsf{Vf}(\mathsf{vk_{sim}}, \sigma_{granted}, \text{“}accepted\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$ or there exists $(\mathsf{tx}_{denied}, \sigma_{denied}) \in \mathsf{chalwindow}$ with $\sigma_{refuse} \in \mathsf{tx}_{denied}.\mathsf{Data}$ and $\Sigma.\mathsf{Vf}(\mathsf{vk_{sim}}, \sigma_{refuse}, \text{“}denied\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$ abort with **SigForge**
  7. If there does not exist $\mathsf{tx}_{reg} \in \mathsf{chalwindow_C}$ with $\sigma_{sig} \in \mathsf{tx}_{reg}.\mathsf{Data}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk_S}, \sigma_{sig}, \mathsf{tx_C}.\mathsf{Data}) = 1$ abort with **ForgeFail**
  8. If there exists any pair $(\mathsf{tx}, \sigma) \in \mathsf{comwindow}$ or $\mathsf{chalwindow}$ such that $(\mathsf{tx}, \sigma) \notin L_{\mathsf{cid}}$, $(\mathsf{tx}_{com}, \sigma_{com}) \notin \mathsf{comwindow}$, $(\mathsf{tx}_{open}, \sigma_{open}) \notin \mathsf{chalwindow}$ abort with **ForgeFail**
  9. Else if there exists $\mathsf{tx}'_{open} \in \mathsf{chalwindow}$ such that there exists $\mathsf{tx}'_{com} \in \mathsf{comwindow}$ where $\mathsf{tx}'_{open}.\mathsf{Data}$ is a valid opening of $\mathsf{tx}'_{com}.\mathsf{Data}$ for a request for the same perm-info, and the distance between tx'_{open} and the commitment it opens is less than or equal to t_{open} abort with **CommitFail**
  10. Else set $\mathsf{perm} = \mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx_C}\|\sigma_\mathsf{C}\|\mathsf{chalwindow_C}\|\mathsf{open}\|\mathsf{comwindow_{req}}\|\mathsf{chalwindow_{req}}\|\mathsf{vk_S}\|\sigma_{perm}$ forward the query, receive $(\mathsf{tx}_{fin}, \sigma_{fin})$ and store in L_{cid} and store in L_{cid} and send perm to $\mathcal{G}_{Perm}$

Figure 12: Simulation of an Honest Request to Generate Permission in Π_{Perm} for a Malicious Server S^* and Party P^*

https://t.me/learningnets

Simulator: $\mathsf{Sim}_{\mathsf{S}^*}$ - Generate Permissions Malicious Request

- **Generate Permission:** Upon query $\texttt{com}$ to $\mathsf{OLedger}$ by P^*

1. Forward $\texttt{com}$ to $\mathsf{OLedger}$, receive $(\mathsf{tx}_{com}, \sigma_{com})$, and store in L_{cid}
2. Upon query $\texttt{open} = \mathsf{perm\text{-}info}\|\mathsf{req}\|\mathsf{S}\|\mathsf{tx}_{\mathsf{C}}; r$ to $\mathsf{OLedger}$ by P^*, forward $\texttt{open}$ to $\mathsf{OLedger}$, receive $(\mathsf{tx}_{open}, \sigma_{open})$ and store in L_{cid}
3. Send $(\texttt{generate permissions}, \mathsf{perm\text{-}info}, \mathsf{S}^*, \mathsf{req})$ to $\mathcal{G}_{Perm}$
4. If $\mathsf{res} = \texttt{acceptance proof}$
   (a) Compute $\sigma_{granted} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sim}}, \text{“}accepted\text{”}\|\mathsf{tx}_{open}.\mathsf{Data})$, query $\mathsf{OLedger}$ with “*accepted*”$\|\mathsf{tx}_{open}.\mathsf{Data}\|\sigma_{granted}$, receive $\mathsf{tx}_{accept}, \sigma_{accept}$ and store in L_{cid}
   (b) Upon query $\mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx}_{\mathsf{C}}\|\sigma_{\mathsf{C}}\|\mathsf{chalwindow}_{\mathsf{C}}\|\mathsf{open}\|\mathsf{comwindow}_{\mathsf{req}}\|\mathsf{chalwindow}_{\mathsf{req}}\|\mathsf{vk}_{\mathsf{S}}\|\sigma_{perm}$ to $\mathsf{OLedger}$ by S^*, if $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{S}}, \sigma_{perm}, \mathsf{tx}_{\mathsf{C}}\|\sigma_{\mathsf{C}}\|\mathsf{chalwindow}_{\mathsf{C}}\|\mathsf{open}\|\mathsf{comwindow}_{\mathsf{req}}\|\mathsf{chalwindow}_{\mathsf{req}})$ $\neq 1$ abort, else
      i. If $(\mathsf{tx}_{accept}, \sigma_{accept}) \notin \mathsf{chalwindow}$ abort with **ForgeFail**
      ii. If there exists $(\mathsf{tx}_{denied}, \sigma_{denied}) \in \mathsf{chalwindow}$ with $\sigma_{refuse} \in \mathsf{tx}_{denied}.\mathsf{Data}$ and $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{sim}}, \sigma_{refuse}, \text{“}denied\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$ abort with **SigForge**
5. Else if $\mathsf{res} = \texttt{denial proof}$
   (a) Compute $\sigma_{refuse} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sim}}, \text{“}denied\text{”}\|\mathsf{tx}_{open}.\mathsf{Data})$, query $\mathsf{OLedger}$ with “*denied*”$\|\mathsf{tx}_{open}.\mathsf{Data}\|\sigma_{refuse}$, receive $\mathsf{tx}_{denied}, \sigma_{denied}$ and store in L_{cid}
   (b) Upon query $\mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx}_{\mathsf{C}}\|\sigma_{\mathsf{C}}\|\mathsf{chalwindow}_{\mathsf{C}}\|\mathsf{open}\|\mathsf{comwindow}_{\mathsf{req}}\|\mathsf{chalwindow}_{\mathsf{req}}\|\mathsf{vk}_{\mathsf{S}}\|\sigma_{perm}$ to $\mathsf{OLedger}$ by S^*, if $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{S}}, \sigma_{perm}, \mathsf{tx}_{\mathsf{C}}\|\sigma_{\mathsf{C}}\|\mathsf{chalwindow}_{\mathsf{C}}\|\mathsf{open}\|\mathsf{comwindow}_{\mathsf{req}}\|\mathsf{chalwindow}_{\mathsf{req}})$ $\neq 1$ abort, else
      i. If $(\mathsf{tx}_{denied}, \sigma_{denied}) \notin \mathsf{chalwindow}$ abort with **ForgeFail**
      ii. If there exists $(\mathsf{tx}_{accept}, \sigma_{accept}) \in \mathsf{chalwindow}$ with $\sigma_{granted} \in \mathsf{tx}_{accept}.\mathsf{Data}$ and $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{sim}}, \sigma_{granted}, \text{“}accepted\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$ abort with **SigForge**
6. Else
   (a) Upon query $\mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx}_{\mathsf{C}}\|\sigma_{\mathsf{C}}\|\mathsf{chalwindow}_{\mathsf{C}}\|\mathsf{open}\|\mathsf{comwindow}_{\mathsf{req}}\|\mathsf{chalwindow}_{\mathsf{req}}\|\mathsf{vk}_{\mathsf{S}}\|\sigma_{perm}$ to $\mathsf{OLedger}$ by S^*, if $|\mathsf{comwindow}_{\mathsf{req}}| \neq t_{open} + \zeta$, $|\mathsf{chalwindow}_{\mathsf{req}}| \neq t_{chal}$, or $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{S}}, \sigma_{perm}, \mathsf{tx}_{\mathsf{C}}\|\sigma_{\mathsf{C}}\|\mathsf{chalwindow}_{\mathsf{C}}\|\mathsf{open}\|\mathsf{comwindow}_{\mathsf{req}}\|\mathsf{chalwindow}_{\mathsf{req}}) \neq 1$ abort
      i. If there exists $(\mathsf{tx}_{accept}, \sigma_{accept}) \in \mathsf{chalwindow}$ with $\sigma_{granted} \in \mathsf{tx}_{accept}.\mathsf{Data}$ and $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{sim}}, \sigma_{granted}, \text{“}accepted\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$ or there exists $(\mathsf{tx}_{denied}, \sigma_{denied}) \in \mathsf{chalwindow}$ with $\sigma_{refuse} \in \mathsf{tx}_{denied}.\mathsf{Data}$ and $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{sim}}, \sigma_{refuse}, \text{“}denied\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$ abort with **SigForge**
7. If there does not exist $\mathsf{tx}_{reg} \in \mathsf{chalwindow}_{\mathsf{C}}$ with $\sigma_{sig} \in \mathsf{tx}_{reg}.\mathsf{Data}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{S}}, \sigma_{sig}, \mathsf{tx}_{\mathsf{C}}.\mathsf{Data}) = 1$ abort with **ForgeFail**
8. If there exists any pair $(\mathsf{tx}, \sigma) \in \mathsf{chalwindow}_{\mathsf{C}} \cup \mathsf{comwindow}_{\mathsf{req}} \cup \mathsf{chalwindow}_{\mathsf{req}}$ such that $(\mathsf{tx}, \sigma) \notin L_{\mathsf{cid}}$, $(\mathsf{tx}_{com}, \sigma_{com}) \notin \mathsf{comwindow}$, $(\mathsf{tx}_{open}, \sigma_{open}) \notin \mathsf{chalwindow}$ abort with **ForgeFail**
9. Else if there exists $\mathsf{tx}'_{open} \in \mathsf{chalwindow}$ such that there exists $\mathsf{tx}'_{com} \in \mathsf{comwindow}$ where $\mathsf{tx}'_{open}.\mathsf{Data}$ is a valid opening of $\mathsf{tx}'_{com}.\mathsf{Data}$ for a request for the same $\mathsf{perm\text{-}info}$, and the distance between tx'_{open} and the commitment it opens is less than or equal to t_{open} abort with **CommitFail**
10. Else set $\mathsf{perm} = \mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx}_{\mathsf{C}}\|\sigma_{\mathsf{C}}\|\mathsf{chalwindow}_{\mathsf{C}}\|\mathsf{open}\|\mathsf{comwindow}_{\mathsf{req}}\|\mathsf{chalwindow}_{\mathsf{req}}\|\mathsf{vk}_{\mathsf{S}}\|\sigma_{perm}$ forward the query, receive $(\mathsf{tx}_{fin}, \sigma_{fin})$ and store in L_{cid} and store in L_{cid} and send perm to $\mathcal{G}_{Perm}$

Figure 13: Simulation of a Malicious Request to Generate Permission in Π_{Perm} for a Malicious Server S^* and Party P^*

https://t.me/learningnets

Simulator: $\mathsf{Sim}_{\mathsf{C}^*}$ **- Register**

- **Register - Server:** Upon receipt of $(\mathtt{registered}, \mathsf{S})$ from $\mathcal{G}_{Perm}$
  1. Set $\mathsf{vk}_{\mathsf{sim}}, \mathsf{sk}_{\mathsf{sim}} \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
  2. Choose cid
  3. Query $\mathsf{OLedger}$ with "$register$"$\|\mathsf{vk}_{\mathsf{sim}}$, receive $(\mathsf{tx}_{\mathsf{S}}, \sigma_{\mathsf{S}})$ and store in L_{cid}
- **Register - Client:** Upon query $\mathsf{perm\text{-}info} = \mathsf{C}^*\|\mathsf{vk}_{\mathsf{C}}\|t_{open}\|t_{chal}\|\mathsf{vk}_{\mathsf{sim}}$ from C^* to $\mathsf{OLedger}$
  1. Forward the query, receive $(\mathsf{tx}_{\mathsf{C}^*}, \sigma_{\mathsf{C}^*})$, store in L_{cid} and receive $(\mathsf{tx}_{\mathsf{C}^*}, \sigma_{\mathsf{C}^*})$ from C^*
  2. Send $(\mathtt{register\ client}, \mathsf{C}^*, \mathsf{S})$ to $\mathcal{G}_{Perm}$, receive $(\mathtt{registration\ request}, \mathsf{C}^*, \mathsf{S})$ from $\mathcal{G}_{Perm}$ and respond with $(\mathtt{client\ perm\text{-}info}, \mathsf{C}^*, \mathsf{perm\text{-}info})$
  3. If this is the first registration request from C^*, compute $\sigma_{sig} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sim}}, \mathsf{tx}_{\mathsf{C}^*}.\mathsf{Data})$, query $\mathsf{OLedger}$ with $(\mathsf{tx}_{\mathsf{C}^*}.\mathsf{Data}\|\sigma_{sig})$, receive $(\mathsf{tx}_{reg}, \sigma_{reg})$ and store in L_{cid}

Figure 14: Simulation of the Register Procedure of Π_{Perm} for a Malicious Client C^*

1. Activate $\mathcal{A}(1^\lambda)$
2. Emulate $\mathbf{Hyb}_0$ for $\mathcal{A}$, posting all queries to $\mathsf{OLedger}$ to $\mathcal{L}$
3. If $\mathcal{A}$ submits a pair $(\mathsf{tx}^*, \sigma^*)$ as a part of the permissions such that $\mathcal{L}.\mathsf{Vf}(\mathsf{tx}^*, \sigma^*) = 1$ but was not a result of a query to $\mathsf{OLedger}$, submit $(\mathsf{tx}^*, \sigma^*)$ to the challenger, else abort

Because $\mathcal{A}$ can distinguish between the two hybrids, we know that the pair $(\mathsf{tx}^*, \sigma^*)$ must verify. Therefore we know that $\mathcal{D}$ wins the unforgeability game for a $\mathsf{SUF\text{-}AUTH}$ ledger with the same non-negligible probability that $\mathcal{A}$ has of distinguishing between the two hybrids. □

Lemma 2. *If Σ is a EUF-CMA secure signature scheme,* $\mathbf{Hyb}_2$ *is indistinguishable from* $\mathbf{Hyb}_1$

Proof. Assume towards a contradiction that there exists an adversary $\mathcal{A}$ such that $|Pr[\mathcal{A}(\mathbf{Hyb}_2) = 1] - Pr[\mathcal{A}(\mathbf{Hyb}_1) = 1]| > \mathsf{negl}(\lambda)$. The only difference between these two hybrids is that the simulator aborts with **SigForge** when there is a signed acceptance or denial transaction that was not computed by the simulator.

Therefore, we can use $\mathcal{A}$ to construct a reduction $\mathcal{D}$ that can win the unforgeability game for a EUF-CMA signature scheme Σ. Define $\mathcal{D}$ as follows:

$\underline{\mathcal{D}(\mathsf{vk})}$:

1. Activate $\mathcal{A}(1^\lambda)$
2. Emulate $\mathbf{Hyb}_1$ for $\mathcal{A}$
3. If $\mathcal{A}$ submits permissions containing a signed acceptance $\sigma^* = \sigma_{granted}$ or denial $\sigma^* = \sigma_{denied}$, send σ^* and the message it signs to the challenger, else abort

Because $\mathcal{A}$ can distinguish between the two hybrids, we know that the signature σ^* will verify. Therefore we know that $\mathcal{D}$ wins the unforgeability game for a EUF-CMA signature scheme with the same non-negligible probability that $\mathcal{A}$ has of distinguishing between the two hybrids. □

Lemma 3. *If* $\mathtt{COM} = (\mathsf{Commit}, \mathsf{Open})$ *is a statistically hiding commitment scheme,* $\mathbf{Hyb}_3$ *is indistinguishable from* $\mathbf{Hyb}_2$

Proof. Because $\mathtt{COM}$ is statistically hiding, we know that an adversary cannot determine the contents of the commitment based on tx_{com}. Therefore, the only way the adversary can post a competing commitment is by guessing which client the request is for. □

Case: Malicious Client Next we consider the case of a malicious client. Towards this, we present the simulator $\mathsf{Sim}_{\mathsf{C}^*}$ (Figures 14, 15). Note that the case of a malicious client covers the case of a malicious party P. This is because C can perform any procedure that P can and more, and with more knowledge. We do still, however, consider the possibility that the permissions were requested by some malicious party P^*.

Proof by Hybrids

https://t.me/learningnets

Simulator: $\mathsf{Sim}_{\mathsf{C}^*}$ - Generate Permission

- **Generate Permission**: Upon query $\mathtt{com}$ to $\mathsf{OLedger}$ by C^* or P^*

1. Forward $\mathtt{com}$ to $\mathsf{OLedger}$, receive $(\mathsf{tx}_{com}, \sigma_{com})$, and store in L_{cid}
2. Upon query $\mathtt{open} = \mathsf{perm\text{-}info}\|\mathsf{req}\|\mathsf{S}\|\mathsf{tx}_{\mathsf{C}^*}; r$ to $\mathsf{OLedger}$ by C^* or P^*, forward $\mathtt{open}$ to $\mathsf{OLedger}$, receive $(\mathsf{tx}_{open}, \sigma_{open})$, and store in L_{cid}
3. Let $\mathsf{chalwindow}_{\mathsf{C}}$ be the t_{chal} blocks after and including tx_{C}
4. Send $(\mathtt{generate\ permission}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req})$ to $\mathcal{G}_{Perm}$
5. If C^* queries $\mathsf{OLedger}$ with "*accepted*"$\|\mathsf{tx}_{open}.\mathsf{Data}\|\sigma_{granted}$ where $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{C}}, \sigma_{granted}, \text{“}accepted\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$, send $(\mathtt{accepted})$ to $\mathcal{G}_{Perm}$
   (a) Forward the query to $\mathsf{OLedger}$, receive $(\mathsf{tx}_{accept}, \sigma_{accept})$, and store in L_{cid}
   (b) Let $\mathsf{comwindow}_{\mathsf{req}}$ be the t_{open} blocks before and ζ blocks after tx_{open} and $\mathsf{chalwindow}_{\mathsf{req}}$ be the blocks from tx_{open} to tx_{accept}
   (c) Compute $\sigma_{perm} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sim}}, \mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathsf{chalwindow}_{\mathsf{C}^*}\|\mathtt{open}\|\mathsf{comwindow}_{\mathsf{req}}\|\mathsf{chalwindow}_{\mathsf{req}})$
6. Else if C^* queries $\mathsf{OLedger}$ with "*denied*"$\|\mathsf{tx}_{open}.\mathsf{Data}\|\sigma_{refuse}$ where $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{C}}, \sigma_{refuse}, \text{“}denied\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$, send $(\mathtt{denied})$ to $\mathcal{G}_{Perm}$
   (a) Forward the query to $\mathsf{OLedger}$, receive $(\mathsf{tx}_{denied}, \sigma_{denied})$, and store in L_{cid}
   (b) Let $\mathsf{comwindow}_{\mathsf{req}}$ be the t_{open} blocks before and ζ blocks after tx_{open} and $\mathsf{chalwindow}_{\mathsf{req}}$ be the blocks from tx_{open} to tx_{denied}
   (c) Compute $\sigma_{perm} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sim}}, \mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathsf{chalwindow}_{\mathsf{C}^*}\|\mathtt{open}\|\mathsf{comwindow}_{\mathsf{req}}\|\mathsf{chalwindow}_{\mathsf{req}})$
7. Else if C^* does nothing, send $(\mathtt{silent})$ to $\mathcal{G}_{Perm}$
   (a) If any party queries $\mathsf{OLedger}$ with $\mathtt{com}'$ and $\mathtt{open}'$ such that $\mathsf{req}' \in \mathtt{open}'$ is a request for the same $\mathsf{perm\text{-}info}$, the distance between the queries is less than t_{open}, and $\mathtt{com}'$ was queried no more than ζ time after $\mathtt{com}$ output $\perp$ and abort
   (b) Else if any party queries $\mathsf{OLedger}$ with $\mathtt{open}'$ such that $\mathtt{open}' \neq \mathtt{open}$ is an opening for $\mathtt{com}$, abort with **BindingFail**
   (c) Let $\mathsf{comwindow}_{\mathsf{req}}$ be the t_{open} blocks before and ζ blocks after tx_{open} and $\mathsf{chalwindow}_{\mathsf{req}}$ be the t_{chal} blocks after tx_{open}
   (d) Compute $\sigma_{perm} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sim}}, \mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathsf{chalwindow}_{\mathsf{C}^*}\|\mathtt{open}\|\mathsf{comwindow}_{\mathsf{req}}\|\mathsf{chalwindow}_{\mathsf{req}})$
8. Set $\mathsf{perm} = \mathsf{tx}_{\mathsf{S}}\|\sigma_{\mathsf{S}}\|\mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathsf{chalwindow}_{\mathsf{C}^*}\|\mathtt{open}\|\mathsf{comwindow}\|\mathsf{chalwindow}\|\mathsf{vk}_{\mathsf{sim}}\|\sigma_{perm}$ query $\mathsf{OLedger}$ with perm, receive $(\mathsf{tx}_{fin}, \sigma_{fin})$, store in L_{cid}, and send perm to $\mathcal{G}_{Perm}$

Figure 15: Simulation of the Generate Permission Procedure of Π_{Perm} for a Malicious Client C^*

https://t.me/learningnets

Simulator: $\mathsf{Sim}_{\mathsf{S}^*,\mathsf{C}^*}$ **- Register**

- **Register - Server:** Upon query ("*register*"$\|\mathsf{vk}_\mathsf{S}$) to the ledger
  1. Send (`register server`, S^*) to $\mathcal{G}_{Perm}$
  2. Forward the query to $\mathsf{OLedger}$ and receive $\mathsf{tx}_{\mathsf{S}^*}, \sigma_{\mathsf{S}^*}$, store this in L_{cid} and forward to S^*
- **Register - Client:** Upon query $\mathsf{perm\text{-}info} = \mathsf{C}^*\|\mathsf{vk}_\mathsf{C}\|t_{open}\|t_{chal}\|\mathsf{vk}_\mathsf{S}$ from C^* to $\mathsf{OLedger}$
  1. Forward the query, receive $\mathsf{tx}_{\mathsf{C}^*}, \sigma_{\mathsf{C}^*}$ and store in L_{cid}
  2. Send (`register client`, $\mathsf{C}^*, \mathsf{S}^*$) to $\mathcal{G}_{Perm}$, receive (`registration request`, $\mathsf{C}^*, \mathsf{S}^*$) from $\mathcal{G}_{Perm}$, and respond with (`client perm-info`, C^*, $\mathsf{perm\text{-}info}$)
  3. Upon query ($\mathsf{tx}_{\mathsf{C}^*}.\mathsf{Data}\|\sigma_{sig}$) to $\mathsf{OLedger}$ by S^*, forward the query to receive ($\mathsf{tx}_{reg}, \sigma_{reg}$) and store in L_{cid}

Figure 16: Simulation of the Register Procedure of Π_{Perm} for a Malicious Server S^* and Client C^*

We prove that the view simulated by $\mathsf{Sim}_{\mathsf{C}^*}$ is indistinguishable from a view of the adversary in the real world through a series of hybrids, starting from the real world protocol and moving step-by-step until we reach the ideal world. By proving that each hybrid is indistinguishable from the last, we will prove that the real and ideal world are indistinguishable to a malicious C^*.

- $\mathbf{Hyb}_0$: The real world protocol
- $\mathbf{Hyb}_1$: This is the same as $\mathbf{Hyb}_0$, except that $\mathsf{Sim}_{\mathsf{C}^*}$ aborts with **BindingFail** when a commitment is opened to a different value than the committed value

Lemma 4. *If* $\mathtt{COM} = (\mathsf{Commit}, \mathsf{Open})$ *is a binding commitment scheme,* $\mathbf{Hyb}_1$ *is indistinguishable from* $\mathbf{Hyb}_0$

Proof. Assume towards a contradiction that there exists an adversary $\mathcal{A}$ such that $|Pr[\mathcal{A}(\mathbf{Hyb}_1) = 1] - Pr[\mathcal{A}(\mathbf{Hyb}_0) = 1]| > \mathsf{negl}(\lambda)$. The only difference between these two hybrids is that in $\mathbf{Hyb}_1$, the simulator aborts when a commitment is opened to two different values. Therefore, we can use $\mathcal{A}$ to construct a reduction $\mathcal{D}$ that can break the binding property of $\mathtt{COM}$. Define $\mathcal{D}$ as follows:

$\underline{\mathcal{D}(1^\lambda)}$:

1. Activate $\mathcal{A}(1^\lambda)$
2. Emulate $\mathbf{Hyb}_0$ for $\mathcal{A}$
3. If $\mathcal{A}$ provides $\mathtt{open}'$ such that $\mathtt{open}'$ is a second opening for $\mathtt{com}$, send ($\mathtt{com}, \mathtt{open}, \mathtt{open}'$) to the challenger, else abort

Because $\mathcal{A}$ is able to distinguish between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$, we know that $\mathtt{open}$ and $\mathtt{open}'$ will be valid openings with the same probability that $\mathcal{A}$ has of distinguishing between the two hybrids. Therefore $\mathcal{D}$ breaks binding with the same non-negligible probability that $\mathcal{A}$ has of distinguishing between the two hybrids. ☐

Case: Malicious Server and Client Now we consider the case of a malicious server colluding with a malicious client.

Proof by Hybrids

We prove that the view simulated by $\mathsf{Sim}_{\mathsf{S}^*,\mathsf{C}^*}$ is indistinguishable from a view of the adversary in the real world through a series of hybrids, starting from the real world protocol and moving step-by-step until we reach the ideal world. By proving that each hybrid is indistinguishable from the last, we will prove that the real and ideal world are indistinguishable to a malicious S^* and C^*.

- $\mathbf{Hyb}_0$: The real world protocol
- $\mathbf{Hyb}_1$: This is the same as $\mathbf{Hyb}_0$ except $\mathsf{Sim}_{\mathsf{S}^*,\mathsf{C}^*}$ aborts with **BindingFail** when a commitment is opened to a different value than the committed value
- $\mathbf{Hyb}_2$: This is the same as $\mathbf{Hyb}_1$ except $\mathsf{Sim}_{\mathsf{S}^*,\mathsf{C}^*}$ aborts with **ForgeFail** if there are any forged transactions in the permissions

Lemma 5. *If* $\mathtt{COM} = (\mathsf{Commit}, \mathsf{Open})$ *is a binding commitment scheme,* $\mathbf{Hyb}_1$ *is indistinguishable from* $\mathbf{Hyb}_0$

https://t.me/learningnets

Simulator: $\mathsf{Sim}_{\mathsf{S}^*,\mathsf{C}^*}$ **- Generate Permission**

- **Generate Permission:** Upon query $\mathtt{com}$ to $\mathsf{OLedger}$ by C^* or P^*
  1. Forward $\mathtt{com}$ to $\mathsf{OLedger}$, receive $(\mathsf{tx}_{com}, \sigma_{com})$ and store in L_{cid}
  2. Upon query $\mathtt{open} = \mathsf{perm\text{-}info}\|\mathsf{req}\|\mathsf{S}\|\mathsf{tx}_\mathsf{C}; r$ to $\mathsf{OLedger}$ from C^* or P^* forward $\mathtt{open}$ to $\mathsf{OLedger}$, receive $(\mathsf{tx}_{open}, \sigma_{open})$ and store in L_{cid}
  3. Send $(\mathtt{generate\ permissions}, \mathsf{perm\text{-}info}, \mathsf{S}^*, \mathsf{req})$ to $\mathcal{G}_{Perm}$
  4. If C^* queries with $(\text{“}accepted\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}\|\sigma_{granted})$ where $\Sigma.\mathsf{Vf}(\mathsf{vk}_\mathsf{C}, \sigma_{granted}, \text{“}accepted\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$, send $(\mathtt{accepted})$ to $\mathcal{G}_{Perm}$
     (a) Forward the query to $\mathsf{OLedger}$ and receive $(\mathsf{tx}_{accept}, \sigma_{accept})$
     (b) Upon query $\mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathsf{chalwindow}_{\mathsf{C}^*}\|\mathtt{open}\|\mathsf{comwindow}_\mathsf{req}\|\mathsf{chalwindow}_\mathsf{req}\|\mathsf{vk}_\mathsf{S}\|\sigma_{perm}$ to $\mathsf{OLedger}$ by S^*, if $\Sigma.\mathsf{Vf}(\mathsf{vk}_\mathsf{S}, \sigma_{perm}, \mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathsf{chalwindow}_\mathsf{C}\|\mathtt{open}\|\mathsf{comwindow}_\mathsf{req}\|\mathsf{chalwindow}_\mathsf{req}) \neq 1$ abort
  5. Else if C^* queries with $(\text{“}denied\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}\|\sigma_{refuse})$ where $\Sigma.\mathsf{Vf}(\mathsf{vk}_\mathsf{C}, \sigma_{refuse}, \text{“}denied\text{”}\|\mathsf{tx}_{open}.\mathsf{Data}) = 1$, send $(\mathtt{denied})$ to $\mathcal{G}_{Perm}$
     (a) Forward the query to $\mathsf{OLedger}$ and receive $(\mathsf{tx}_{denied}, \sigma_{denied})$
     (b) Upon query $\mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathsf{chalwindow}_{\mathsf{C}^*}\|\mathtt{open}\|\mathsf{comwindow}_\mathsf{req}\|\mathsf{chalwindow}_\mathsf{req}\|\mathsf{vk}_\mathsf{S}\|\sigma_{perm}$ to $\mathsf{OLedger}$ by S^*, if $\Sigma.\mathsf{Vf}(\mathsf{vk}_\mathsf{S}, \sigma_{perm}, \mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathsf{chalwindow}_\mathsf{C}\|\mathtt{open}\|\mathsf{comwindow}_\mathsf{req}\|\mathsf{chalwindow}_\mathsf{req}) \neq 1$ abort
  6. Else if C^* does nothing, send $(\mathtt{silent})$ to $\mathcal{G}_{Perm}$
     (a) If any party queries $\mathsf{OLedger}$ with $\mathtt{com}'$ and $\mathtt{open}'$ such that $\mathsf{req}' \in \mathtt{open}'$ is a request for the same $\mathsf{perm\text{-}info}$, the distance between the queries is less than t_{open}, and $\mathtt{com}'$ was queried no more than ζ time after $\mathtt{com}$ output $\perp$ and abort
     (b) Else if any party queries $\mathsf{OLedger}$ with $\mathtt{open}'$ such that $\mathtt{open}' \neq \mathtt{open}$ is an opening for $\mathtt{com}$, abort with **BindingFail**
     (c) Upon query $\mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathsf{chalwindow}_{\mathsf{C}^*}\|\mathtt{open}\|\mathsf{comwindow}_\mathsf{req}\|\mathsf{chalwindow}_\mathsf{req}\|\mathsf{vk}_\mathsf{S}\|\sigma_{perm}$ to $\mathsf{OLedger}$ by S^*, if $|\mathsf{comwindow}_\mathsf{req}| \neq t_{open} + \zeta$, $|\mathsf{chalwindow}_\mathsf{req}| \neq t_{chal}$, or $\Sigma.\mathsf{Vf}(\mathsf{vk}_\mathsf{S}, \sigma_{perm}, \mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathsf{chalwindow}_\mathsf{C}\|\mathtt{open}\|\mathsf{comwindow}_\mathsf{req}\|\mathsf{chalwindow}_\mathsf{req}) \neq 1$ abort
  7. If there does not exist $\mathsf{tx}_{reg} \in \mathsf{chalwindow}_\mathsf{C}$ with $\sigma_{sig} \in \mathsf{tx}_{reg}.\mathsf{Data}$ such that $\Sigma.\mathsf{Vf}(\mathsf{vk}_\mathsf{S}, \sigma_{sig}, \mathsf{tx}_\mathsf{C}.\mathsf{Data}) = 1$ abort with **ForgeFail**
  8. If there exists any pair $(\mathsf{tx}, \sigma) \in \mathsf{chalwindow}_{\mathsf{C}^*} \cup \mathsf{comwindow}_\mathsf{req} \cup \mathsf{chalwindow}_\mathsf{req}$ with $(\mathsf{tx}, \sigma) \notin L_\mathsf{cid}$, $(\mathsf{tx}_{com}, \sigma_{com}) \notin \mathsf{comwindow}$, $(\mathsf{tx}_{open}, \sigma_{open}) \notin \mathsf{chalwindow}$, or $(\mathsf{tx}_{denied}, \sigma_{denied}) \notin \mathsf{chalwindow}$ abort with **ForgeFail**
  9. Else set $\mathsf{perm} = \mathsf{tx}_{\mathsf{S}^*}\|\sigma_{\mathsf{S}^*}\|\mathsf{tx}_{\mathsf{C}^*}\|\sigma_{\mathsf{C}^*}\|\mathtt{open}\|\mathsf{comwindow}\|\mathsf{chalwindow}\|\mathsf{vk}_\mathsf{S}\|\sigma_{perm}$ forward the query, receive $(\mathsf{tx}_{fin}, \sigma_{fin})$ and store in L_cid and store in L_cid and send perm to $\mathcal{G}_{Perm}$

Figure 17: Simulation of the Generate Permission Procedure of Π_{Perm} for a Malicious Server S^*, Client C^*, and Party P^*

https://t.me/learningnets

Simulator: Sim_H - Register

- **Registration - Server:** Upon receipt of $(\texttt{registered}, \mathsf{S})$ from $\mathcal{G}_{Perm}$
  1. Set $\mathsf{vk}_\mathsf{S}, \mathsf{sk}_\mathsf{S} \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
  2. Choose cid
  3. Query $\mathsf{OLedger}$ with "$register$"$\|\mathsf{vk}_\mathsf{S}\|\mathsf{cid}$, receive $(\mathsf{tx}_\mathsf{S}, \sigma_\mathsf{S})$ and store in L_cid
- **Registration - Client:** Upon receipt of $(\texttt{register client}, \mathsf{C}, \mathsf{S})$ from $\mathcal{G}_{Perm}$
  1. Set $(\mathsf{vk}_\mathsf{C}, \mathsf{sk}_\mathsf{C}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
  2. Choose t_{open}, t_{chal} as the time to open a commitment and challenge a request respectively
  3. Set $\mathsf{perm\text{-}info} = (\mathsf{C}, \mathsf{vk}_\mathsf{C}, t_{open}, t_{chal}, \mathsf{vk}_\mathsf{S})$ and send $(\texttt{client perm-info}, \mathsf{C}, \mathsf{perm\text{-}info})$
  4. Query $\mathsf{OLedger}$ with $\mathsf{perm\text{-}info}$, receive $\mathsf{tx}_\mathsf{C}, \sigma_\mathsf{C}$, store in L_cid
  5. Compute $\sigma_{sig} = \Sigma.\mathsf{Sign}(\mathsf{sk}_\mathsf{S}, \mathsf{tx}_\mathsf{C}.\mathsf{Data})$, query $\mathsf{OLedger}$ with $(\mathsf{tx}_\mathsf{C}.\mathsf{Data}\|\sigma_{sig})$, receive $(\mathsf{tx}_{reg}, \sigma_{reg})$ and store in L_cid

Figure 18: Simulation of the Registration Procedure of Π_{Perm} for Honest Client C, Server S, and Party P

Proof. Follows from the proof of Lemma 4. □

Lemma 6. *If* $\mathsf{OLedger}$ *is realized by a* $\mathsf{SUF\text{-}AUTH}$ *secure ledger* $\mathcal{L}$*,* $\mathbf{Hyb}_2$ *is indistinguishable from* $\mathbf{Hyb}_1$

Proof. Follows from the proof of Lemma 1. □

Case: All Honest Parties Finally, we consider the case where all parties are honest, and show the simulator Sim_H for this case.

Lemma 7. *The view generated by* Sim_H *is indistinguishable from the view generated by honest parties in the real world*

Proof. Sim_H honestly generates signing keys on behalf of S and C, honestly signs the correct messages based on the commands of $\mathcal{Z}$, honestly commits to the request and posts the opening, posts the correct information to the ledger, and honestly generates the permission. Therefore, the two views are statistically indistinguishable. □

Verifying Permission Next we prove that verification of permission in the real world is indistinguishable from verification of permission in the ideal world using the predicate $\mathsf{VerifyPerm}$ by presenting the simulator Sim.

Lemma 8. *The probability that* Sim_{vf} *aborts with* **VerifyFail** *is* $\mathsf{negl}(\lambda)$

Proof. Sim_{vf} only aborts with **VerifyFail** in two cases:

- **Case:** $\mathsf{res} = \texttt{accepted}$ **and** $b_{ver} = 0$
  In this case, $\mathcal{G}_{Perm}$ accepts the permission while $\mathsf{VerifyPerm}$ rejects the permission. $\mathcal{G}_{Perm}$ will accept permission only if the permission was generated through $\mathcal{G}_{Perm}$ and accepted by the client, or if the permission is generated through silence and $\mathsf{VerifyPerm}$ outputs 1. In the case of silence permission, $\mathcal{G}_{Perm}$ defers to $\mathsf{VerifyPerm}$, therefore the probability that the output differs is 0. In the case of accepted permission, an adversary would need to either forge ledger blocks to remove the acceptance transaction, or forge a denial signature on behalf of the client, which we have proved happens with negligible probability in the proofs of Lemmas 1 and 2 respectively.
- **Case:** $\mathsf{res} = \texttt{denied}$ **and** $b_{ver} = 1$
  In this case, $\mathcal{G}_{Perm}$ denies the permission while $\mathsf{VerifyPerm}$ accepts. $\mathcal{G}_{Perm}$ will deny the permission only if the permission was not generated through $\mathcal{G}_{Perm}$, was generated through $\mathcal{G}_{Perm}$ but denied by the client, or was generated through silence and $\mathsf{VerifyPerm}$ output 0. Again in the case of silence $\mathcal{G}_{Perm}$ defers to $\mathsf{VerifyPerm}$, therefore the probability that the output differs is 0. If the permission was not generated through $\mathcal{G}_{Perm}$, an adversary must have forged blocks from the ledger, as we have shown that

https://t.me/learningnets

Simulator: Sim_H - Generate Permission

- **Generate Permission** Upon receipt of ($\texttt{permission request}$, perm-info) from $\mathcal{G}_{Perm}$

1. Upon receipt of ($\mathsf{res}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}, t_{elapse}$) from $\mathcal{G}_{Perm}$
2. Query OLedger with $\mathtt{com}$ where $(\mathtt{com}, \mathtt{open}) = \mathsf{Commit}(\mathsf{perm\text{-}info} \| \mathsf{req} \| \mathsf{S} \| \mathsf{tx_C})$, receive $\mathsf{tx}_{com}, \sigma_{com}$, and store in L_{cid}
3. After time t_{elapse}, query OLedger with ($\mathtt{open}$), and store $\mathsf{tx}_{open}, \sigma_{open}$ in L_{cid}
4. Let $\mathsf{chalwindow_C}$ be the t_{chal} blocks after and including $\mathsf{tx_C}$
5. If $\mathsf{res} = \texttt{acceptance proof}$
   (a) Compute $\sigma_{granted} = \Sigma.\mathsf{Sign}(\mathsf{sk_C}, \text{“}accepted\text{”} \| \mathsf{tx}_{open}.\mathsf{Data})$, query OLedger with “*accepted*”$\|\mathsf{tx}_{open}.\mathsf{Data}\|\sigma_{granted}$, receive $\mathsf{tx}_{accepted}, \sigma_{accepted}$, and store in L_{cid}
   (b) Let $\mathsf{comwindow_{req}}$ be the t_{open} blocks before and ζ blocks after tx_{open} and $\mathsf{chalwindow_{req}}$ be the blocks from tx_{open} to tx_{accept}
6. If $\mathsf{res} = \texttt{denial proof}$
   (a) Compute $\sigma_{refuse} = \Sigma.\mathsf{Sign}(\mathsf{sk_C}, \text{“}denied\text{”} \| \mathsf{tx}_{open}.\mathsf{Data})$, query OLedger with “*denied*”$\|\mathsf{tx}_{open}.\mathsf{Data}\|\sigma_{refuse}$, receive $\mathsf{tx}_{denied}, \sigma_{denied}$, and store in L_{cid}
   (b) Let $\mathsf{comwindow_{req}}$ be the t_{open} blocks before and ζ blocks after tx_{open} and $\mathsf{chalwindow_{req}}$ be the blocks from tx_{open} to tx_{denied}
7. Else
   (a) Let $\mathsf{comwindow_{req}}$ be the t_{open} blocks before and ζ blocks after tx_{open} and $\mathsf{chalwindow_{req}}$ be the t_{chal} blocks after tx_{open}
8. Compute $\sigma_{perm} = \Sigma.\mathsf{Sign}(\mathsf{sk_S}, \mathsf{tx_C} \| \sigma_\mathsf{C} \| \mathsf{chalwindow_C} \| \mathsf{open} \| \mathsf{comwindow_{req}} \| \mathsf{chalwindow_{req}} \| \mathsf{vk_S})$
9. Set $\mathsf{perm} = \mathsf{tx_S} \| \sigma_\mathsf{S} \| \mathsf{tx_C} \| \sigma_\mathsf{C} \| \mathsf{chalwindow_C} \| \mathsf{open} \| \mathsf{comwindow} \| \mathsf{chalwindow} \| \mathsf{vk_S} \| \sigma_{perm}$, query OLedger with perm, receive $(\mathsf{tx}_{fin}, \sigma_{fin})$ store in L_{cid}, and send perm to $\mathcal{G}_{Perm}$

Figure 19: Simulation of the Generate Permission Procedure of Π_{Perm} for Honest Client C, Server S, and Party P

Simulator: Sim_{vf} - Verify Permission

- **Verify Permission:** Upon receipt of ($\mathsf{res}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}, \mathsf{perm}$) from $\mathcal{G}_{Perm}$

1. Run $\mathsf{VerifyPerm}(\mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}, \mathsf{perm}) = b_{ver}$
2. If $\mathsf{res} = \texttt{accepted}$ and $b_{ver} = 0$ or $\mathsf{res} = \texttt{denied}$ or $\texttt{not verified}$ and $b_{ver} = 1$ abort with **VerifyFail**
3. Else output b_{ver}

Figure 20: Simulation of the Verification of Permission VerifyPerm

https://t.me/learningnets

Functionality: $\mathcal{F}_{SecRec}$

Participants: The Client, the Cloud, the adversary $\mathcal{A}$, and some party P
Variables: $\mathbb{T}_{\mathcal{F}}$ a table of keys indexed by identity
External Functionalities: $\mathcal{G}_{Perm}$ the ideal permissions functionality

Procedures:

- **Store**
  1. Upon receiving $(\texttt{store}, \mathsf{perm\text{-}info}, \mathsf{Cloud}, s)$ from Client, send $(\texttt{storage request}, \mathsf{perm\text{-}info})$ to Cloud and $\mathcal{A}$.
  2. If $\mathbb{T}_{\mathcal{F}}[\mathsf{perm\text{-}info}] \neq \perp$, output $\perp$ to Client and send $(\texttt{existing entry}, \mathsf{perm\text{-}info})$ to Cloud and $\mathcal{A}$
  3. If Cloud is corrupt:
     (a) Upon receipt of $(\mathsf{res}, \mathsf{perm\text{-}info})$ from Cloud, if $\mathsf{res} = \texttt{denied}$ send $\perp$ to Client
     (b) Else store $\mathbb{T}_{\mathcal{F}}[\mathsf{perm\text{-}info}] = (s, \perp)$, send $(\texttt{stored}, \mathsf{perm\text{-}info})$ to Client, Cloud, and $\mathcal{A}$
  4. Else:
     (a) Store $\mathbb{T}_{\mathcal{F}}[\mathsf{perm\text{-}info}] = (s, \perp)$, send $(\texttt{stored}, \mathsf{perm\text{-}info})$ to Client, Cloud, and $\mathcal{A}$
- **Remove**
  1. Upon receipt of $(\texttt{remove}, \mathsf{perm\text{-}info})$ from Client, set $\mathbb{T}_{\mathcal{F}}[\mathsf{perm\text{-}info}] = \perp$ and send $(\texttt{removed}, \mathsf{perm\text{-}info})$ to Cloud and $\mathcal{A}$
- **Retrieve**
  1. Upon receipt of $(\texttt{retrieve}, \mathsf{perm\text{-}info})$ from Client, if $\mathbb{T}_{\mathcal{F}}[\mathsf{perm\text{-}info}] \neq \perp$, send $(\texttt{retrieve}, \mathsf{perm\text{-}info})$ to Cloud and $\mathcal{A}$. If a retrieval request is received from any party other than Client, send $(\texttt{retrieval denied}, \mathsf{perm\text{-}info})$ to Cloud and $\mathcal{A}$
  2. If $\mathbb{T}_{\mathcal{F}}[\mathsf{perm\text{-}info}] = (s, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$ for $\mathsf{req}\|\mathsf{res}\|\mathsf{perm} \neq \perp$ send $(\texttt{recovered}, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$ to Client
  3. Else send (s) to Client and $(\texttt{retrieved}, \mathsf{perm\text{-}info})$ to Cloud and $\mathcal{A}$
- **Recover**
  1. Upon receiving $(\texttt{recover}, \mathsf{perm\text{-}info})$ from a party P, if $\mathbb{T}_{\mathcal{F}}[\mathsf{perm\text{-}info}] \neq \perp$, send $(\texttt{recover}, \mathsf{perm\text{-}info})$ to Client, Cloud, and $\mathcal{A}$
  2. Upon receiving $(\texttt{recover}, \mathsf{perm\text{-}info}, \mathsf{req}, \mathsf{perm})$ from $\mathcal{A}$ send $(\texttt{verify permission}, \mathsf{perm\text{-}info}, \mathsf{Cloud}, \mathsf{req}, \mathsf{perm})$ to $\mathcal{G}_{Perm}$
  3. Upon receipt of $(\mathsf{res}, \mathsf{perm\text{-}info}, \mathsf{Cloud}, \mathsf{req}\|\mathsf{perm})$ from $\mathcal{G}_{Perm}$, if $\mathsf{res} \neq \texttt{accepted}$ output $\perp$ to P
  4. Else Send $(s, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$ to P, set $\mathbb{T}_{\mathcal{F}}[\mathsf{perm\text{-}info}] = (s, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$, and send $(\texttt{recovery accepted}, \mathsf{perm\text{-}info}, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$ to Client, Cloud, $\mathcal{A}$

Figure 21: $\mathcal{F}_{SecRec}$ The Ideal Functionality for Secret Recovery

permission generation can be simulated in all cases, and we know that blocks can only be forged with negligible probability in the proof of Lemma 1. In the case of denied permission, an adversary would again have to either forge ledger blocks to remove the denial transaction, or forge an acceptance signature on behalf of the client. We know this happens with negligible probability from the proofs of Lemmas 1 and 2 respectively.

Therefore, in both cases, the probability that Sim_{vf} aborts with **VerifyFail** is negligible. □

6 Credential-less Secret Recovery

In this section we present our definition of secret recovery, followed by our protocol realizing our definition and the proof of security.

https://t.me/learningnets

6.1 Definition of Secret Recovery

In Figure 21 we present the ideal functionality for secret recovery $\mathcal{F}_{SecRec}$. With this functionality we capture a client storing a secret with a cloud, such that this secret can be recovered only using permissions obtained through $\mathcal{G}_{Perm}$. The cloud learns nothing about the secret during storage, retrieval, or recovery. Further, a client is able to request removal from the secret recovery system, at which point the secret will no longer be stored.

6.2 Secret Recovery Protocol

Next we present the secret recovery protocol Π_{SecRec} (Figures 22, 23), as well as the program run by $\mathcal{G}_{att}$ (Figure 24), and provide a sketch of the proof that Π_{SecRec} realizes $\mathcal{F}_{SecRec}$ in the $\mathcal{G}_{Perm}, \mathcal{G}_{att}$-hybrid world.

6.3 Instantiating Protocol Π_{SecRec}.

In order to implement Π_{SecRec}, we need an INT-CTX and IND-CPA secure symmetric key encryption scheme, an IND-CPA secure public key encryption scheme, and an EUF-CMA secure signature scheme.

INT-CTX and IND-CPA can be achieved using an IND-CPA secure encryption scheme and a message authentication code (MAC) via encrypt-then-MAC [BN00]. For the encryption scheme we suggest AES [Dwo23] and for the MAC we suggest HMAC [BCK96].

For our public key encryption, we suggest ElGamal [Gam84]. Encryption is done by the TEE upon recovery only, *decryption is done by the client once* during recovery only. Decryption is performed by the client once, upon receiving the recovered secret. For signatures, we again we will use the slightly modified ElGamal signature scheme [Gam84, Bon11]. The client need only sign in the case of removing their stored secret or upon retrieval. The cloud signs nothing, only verifies the signatures of the client.

6.4 Security Proofs

Finally we present our theorem that Π_{SecRec} realizes $\mathcal{F}_{SecRec}$ in the $\mathcal{G}_{Perm}, \mathcal{G}_{att}$-hybrid world.

Theorem 2. *If* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is an INT-CTX and IND-CPA secure symmetric key encryption scheme,* $\mathcal{G}_{att}$ *is parameterized by an EUF-CMA signature scheme, the DDH assumption holds in group* G *of prime order* q*,* $\Pi_{pub} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is an IND-CPA secure public key encryption scheme, and* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$ *is an EUF-CMA secure signature scheme, then* Π_{SecRec} *realizes the ideal functionality* $\mathcal{F}_{SecRec}$ *in the* $(\mathcal{G}_{att}, \mathcal{G}_{Perm})$ *- hybrid model*

To prove Theorem 2, we consider four cases: a malicious Cloud^*, a malicious Client^*, both malicious Cloud^* and Client^*, and both honest Cloud and Client. We make use of the same techniques of simulating a secure enclave presented by Pass et al. [PST17]. Specifically, we provide our simulator with a backdoor to the TEE that allows the simulator to obtain signatures on values that are not the true output of the program.

Case: Malicious Cloud^* In Figure 25 we present the simulator in the case of a malicious Cloud^* and prove through a series of hybrids that the view generated by $\mathsf{Sim}_{\mathsf{Cloud}^*}$ is indistinguishable from the view generated in the real world.

Proof by Hybrids

We prove indistinguishability through a series of hybrids:

- $\mathbf{Hyb}_0$: The real world protocol
- $\mathbf{Hyb}_1$: This is the same as $\mathbf{Hyb}_0$ except that a random key is used for retK instead of the result of the Diffie Hellman Key Exchange
- $\mathbf{Hyb}_2$: This is the same as $\mathbf{Hyb}_1$ except that 0 is stored instead of the actual secret and the simulator aborts with **CTXFail** when the ciphertext retrieved is not the ciphertext that was stored
- $\mathbf{Hyb}_3$: This is the same as $\mathbf{Hyb}_2$ except that the simulator aborts with **AttestFail** if Cloud^* does not make the correct calls to $\mathcal{G}_{att}$
- $\mathbf{Hyb}_4$: This is the same as $\mathbf{Hyb}_3$ except that the simulator aborts with **SigForge** if Cloud^* makes a remove call to $\mathcal{G}_{att}$ using a signature that the simulator did not compute

https://t.me/learningnets

Protocol: Π_{SecRec} - Set Up, Store, Manage Permissions, and Remove

- **Set Up**
  - Cloud: Upon receipt of ($\texttt{register server}$, Cloud) from $\mathcal{Z}$
    1. Send ($\texttt{register server}$, Cloud) to $\mathcal{G}_{Perm}$
- **Store**
  - Client: Upon receipt of ($\texttt{store}$, Client, s, Cloud) from $\mathcal{Z}$
    1. Send ($\texttt{register client}$, Client, Cloud) to $\mathcal{G}_{Perm}$ and receive (perm-info, Cloud)
    2. Set $(\mathsf{vk}_{\mathsf{Client}}, \mathsf{sk}_{\mathsf{Client}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
    3. Get $\mathsf{mvk} = \mathcal{G}_{att}.getpk()$
    4. Sample $a \xleftarrow{\$} \mathbb{Z}_q$ and let $A = g^a$
    5. Send (A, perm-info, $\mathsf{vk}_{\mathsf{Client}}$) to Cloud
  - Cloud: Upon receipt of (A, perm-info, $\mathsf{vk}_{\mathsf{Client}}$) from Client
    1. Let $\mathsf{eid} = \mathcal{G}_{att}.install(\text{perm-info}, \mathsf{prog}_{SecRec})$
    2. Let $(\mathsf{eid}, \mathsf{prog}_{SecRec}, B, \mathsf{vk}_{ko}, A, \sigma_{att}) = \mathcal{G}_{att}.resume(\mathsf{eid}, (\text{“store”}, A, \mathsf{vk}_{\mathsf{Client}}))$
    3. Send $(\mathsf{eid}, \mathsf{prog}_{SecRec}, B, \mathsf{vk}_{ko}, A, \sigma_{att})$ to Client
  - Client : Upon receiving $(\mathsf{eid}, \mathsf{prog}_{SecRec}, B, \mathsf{vk}_{\mathsf{Client}}, A, \sigma_{att})$ from Cloud
    1. If $\Sigma_{att}.\mathsf{Vf}(\mathsf{mvk}, \mathsf{eid}\|\mathsf{prog}_{SecRec}\|B\|\mathsf{vk}_{\mathsf{Client}}\|A, \sigma_{att}) \neq 1$, output $\perp$ and abort
    2. Else store $\mathsf{retK} = B^a$
    3. Compute $c = \Pi.\mathsf{Enc}(\mathsf{retK}, (\text{perm-info}, s, \perp))$ and send c to Cloud
  - Cloud: Upon receipt of c from Client
    1. Let $(b_{att}, \sigma_{att}) = \mathcal{G}_{att}.resume(\mathsf{eid}, (\text{“verify ciphertext”}, \text{perm-info}, c))$
    2. If $b_{att} \neq 1$, output $\perp$ and abort
    3. Else set $\mathbb{T}[\text{perm-info}] = (\mathsf{eid}, \mathsf{vk}_{\mathsf{Client}}, c, \perp)$
- **Manage Permissions** (Run continuously by Client upon registration)
  - Client : Upon receipt of any ($\texttt{generate permission}$, perm-info, Cloud, req) from $\mathcal{G}_{Perm}$ such that Client did not request the permission
    1. If $\mathcal{Z}$ sends ($\texttt{accepted}$, req), send ($\texttt{accepted}$) to $\mathcal{G}_{Perm}$
    2. Else if $\mathcal{Z}$ sends ($\texttt{denied}$, req), send ($\texttt{denied}$) to $\mathcal{G}_{Perm}$
    3. Else send ($\texttt{silent}$) to $\mathcal{G}_{Perm}$
- **Remove**
  - Client : Upon receipt of ($\texttt{remove}$, perm-info) from $\mathcal{Z}$
    1. Compute $\sigma_{remove} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{Client}}, \text{“remove”}\|\text{perm-info})$
    2. Send ($\texttt{remove}$, perm-info, σ_{remove}) to Cloud
  - Cloud : Upon receipt of ($\texttt{remove}$, perm-info) from Client
    1. If $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{Client}}, \sigma_{remove}, \text{“remove”}\|\text{perm-info}) \neq 1$ abort
    2. Else let $(\text{“removed”}, \text{perm-info}, \sigma_{remove}, \sigma_{att}) = \mathcal{G}_{att}.resume(\mathsf{eid}, (\text{“remove”}, \text{perm-info}, \sigma_{remove})$
    3. Set $\mathbb{T}[\text{perm-info}] = \perp$
    4. Send (“removed”, perm-info, σ_{att}) to Client
  - Client : Upon receipt of (“removed”, perm-info, $\sigma_{remove}, \sigma_{att}$) from Cloud
    1. If $\Sigma_{att}.\mathsf{Vf}(\mathsf{mvk}, \sigma_{att}, \text{“removed”}\|\text{perm-info}\|\sigma_{remove}) \neq 1$ abort

Figure 22: Set Up, Store, Manage Permissions, and Remove Procedures for the Secret Recovery Protocol Π_{SecRec}

Lemma 9. *If the DDH assumptions holds in the group G of prime order q, $\mathbf{Hyb}_1$ is indistinguishable from $\mathbf{Hyb}_0$*

Proof. Towards a contradiction assume that there exists an adversary $\mathcal{A}$ such that $|Pr[\mathcal{A}(\mathbf{Hyb}_1) = 1] - Pr[\mathcal{A}(\mathbf{Hyb}_0) = 1]| > \mathsf{negl}(\lambda)$. Then we can construct a reduction $\mathcal{D}$ that can distinguish between a random tuple and a DDH tuple with the same non-negligible probability. Define $\mathcal{D}$ as follows:

$\underline{\mathcal{D}(g, A, B, C):}$

https://t.me/learningnets

Protocol: Π_{SecRec} - Retrieve and Recover

- **Retrieve**
  - Client : Upon receipt of ($\texttt{retrieve}$, perm-info) from $\mathcal{Z}$
    1. Compute $\sigma_{ret} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{Client}}, \texttt{retrieve}\|\text{perm-info})$
    2. Send ($\texttt{retrieve}$, perm-info, σ_{ret}) to Cloud
  - Cloud: Upon receipt of ($\texttt{retrieve}$, perm-info) from Client
    1. Get $\mathbb{T}$[perm-info] = (eid, $\mathsf{vk}_{\mathsf{Client}}$, c, req‖res‖perm)
    2. If $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{Client}}, \sigma_{ret}, \texttt{retrieve}\|\text{perm-info}) \neq 1$ abort
    3. Else if req‖res‖perm $\neq \perp$ send ($\texttt{recovered}$, req‖res‖perm) to Client
    4. Else send c to Client
  - Client: Upon receipt of c from Cloud
    1. Compute (perm-info′, s, req‖res‖perm) = $\Pi.\mathsf{Dec}(\mathsf{retK}, c)$
    2. If perm-info′ $\neq$ perm-info output $\perp$ and abort
    3. Else if req‖res‖perm $\neq \perp$ output $\perp$ and abort
- **Recover**
  - Client : Upon receipt of ($\texttt{recover}$, perm-info) from $\mathcal{Z}$
    1. Let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Pi_{pub}.\mathsf{Gen}(1^{\lambda})$
    2. Let req = "*recover*"‖pk
    3. Send ($\texttt{generate permission}$, perm-info, Cloud, req) to $\mathcal{G}_{Perm}$
    4. Upon receipt of ($\texttt{generate permission}$, perm-info, Cloud, req) from $\mathcal{G}_{Perm}$, if the request was not made by Client defer to **Manage Permissions**, else
    5. If $\mathcal{Z}$ sends ($\texttt{accepted}$, req) send ($\texttt{accepted}$) to $\mathcal{G}_{Perm}$
    6. Else if $\mathcal{Z}$ sends ($\texttt{denied}$, req) send ($\texttt{denied}$) to $\mathcal{G}_{Perm}$
    7. Else send ($\texttt{silent}$) to $\mathcal{G}_{Perm}$
    8. Upon receipt of ($\texttt{permission}$, perm-info, Cloud, req‖res‖perm) from $\mathcal{G}_{Perm}$, send req‖res‖perm to Cloud
  - Cloud : Upon receipt of req‖res‖perm from Client
    1. Let $(\mathsf{eid}, c_{fin}, \sigma_{att}) = \mathcal{G}_{att}.\mathit{resume}(\mathsf{eid}, (\text{“recover”}, \mathsf{pk}, c, \mathsf{req}, \mathsf{perm}))$
    2. Set $\mathbb{T}$[perm-info] = (eid, $\mathsf{vk}_{\mathsf{Client}}$, c_{fin}, req‖res‖perm)
    3. Send (eid, c_{fin}, σ_{att}) to Client
  - Client : Upon receipt of (eid, c_{fin}, σ_{att}) from Cloud
    1. Get mvk = $\mathcal{G}_{att}.\mathit{getpk}()$
    2. If $\Sigma_{att}.\mathsf{Vf}(\mathsf{mvk}, \sigma_{att}, \mathsf{outp}) \neq 1$ output $\perp$ and abort
    3. Compute (perm-info, s, req′‖res′‖perm′) = $\Pi_{pub}.\mathsf{Dec}(\mathsf{sk}, c_{fin})$
    4. If req′ $\neq$ req, res′ $\neq$ res, or perm′ $\neq$ perm output $\perp$ and abort

Figure 23: Retrieve and Recover Procedures for the Secret Recovery Protocol Π_{SecRec}

https://t.me/learningnets

Program: prog_{SecRec}

- On input(“store”, $A, \mathsf{vk}_{\mathsf{Client}}$):
  1. Let $b \xleftarrow{\$} \mathbb{Z}_q$
  2. Let $B = g^b$
  3. Let $\mathsf{retK} = A^b$
  4. Output $(\mathsf{eid}, \mathsf{prog}_{SecRec}, B, \mathsf{vk}_{\mathsf{Client}}, A)$
- On input(“verify ciphertext”, $\mathsf{perm\text{-}info}, c$):
  1. Compute $(\mathsf{perm\text{-}info}', s, \mathsf{req}\|\mathsf{res}\|\mathsf{perm}) = \Pi.\mathsf{Dec}(\mathsf{retK}, c)$
  2. If $\mathsf{perm\text{-}info}' = \mathsf{perm\text{-}info}$ and $\mathsf{req}\|\mathsf{res}\|\mathsf{perm} = \perp$ output 1, else output 0
- On input(“remove”, $\mathsf{perm\text{-}info}, \sigma_{remove}$)
  1. If $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{Client}}, \sigma_{remove}, \text{“remove”}\|\mathsf{perm\text{-}info}) \neq 1$ output $\perp$
  2. Set $\mathsf{retK} = \perp$, $b = \perp$, $B = \perp$, and $A = \perp$
  3. Output “removed” $\|\mathsf{perm\text{-}info}\|\sigma_{remove}$
- On input(“recover”, $\mathsf{pk}, c, \mathsf{req}, \mathsf{perm}$):
  1. If $\mathsf{VerifyPerm}(\mathsf{perm\text{-}info}, \mathsf{Cloud}, \mathsf{req}, \mathsf{perm}) = 0$ or $\mathsf{req} \neq \text{“}recover\text{”}\|\mathsf{pk}$ output $\perp$ and abort
  2. Else $(\mathsf{perm\text{-}info}', s, \mathsf{req}'\|\mathsf{res}'\|\mathsf{perm}') = \Pi.\mathsf{Dec}(\mathsf{retK}, c)$
  3. If $\mathsf{perm\text{-}info}' \neq \mathsf{perm\text{-}info}$ or $\mathsf{req}'\|\mathsf{res}'\|\mathsf{perm}' \neq \perp$ output $\perp$ and abort
  4. $c_{fin} = \Pi_{pub}.\mathsf{Enc}(\mathsf{pk}, (\mathsf{perm\text{-}info}, s, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$
  5. Output (eid, c_{fin})

Figure 24: The Program Run by $\mathcal{G}_{att}$ for Secret Recovery

1. Activate $\mathcal{A}(1^\lambda)$
2. Emulate $\mathbf{Hyb}_0$ for $\mathcal{A}$ using A in step 2 of the storage procedure, B in step 5 of the storage procedure[10] and C as retK
3. Output whatever $\mathcal{A}$ outputs

The only difference between the two hybrids is whether retK is random or g^{ab}. Therefore, if C is random this is exactly $\mathbf{Hyb}_1$ and if $C = g^{ab}$ this is exactly $\mathbf{Hyb}_0$. Therefore $\mathcal{D}$ wins the DDH game with the same non-negligible probability that $\mathcal{A}$ has of distinguishing between the two hybrids. □

Lemma 10. *If $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is INT-CTXT and IND-CPA secure and $\Pi_{pub} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CPA secure, $\mathbf{Hyb}_2$ is indistinguishable from $\mathbf{Hyb}_1$*

Proof. Towards a contradiction assume that there exists an adversary $\mathcal{A}$ such that $|Pr[\mathcal{A}(\mathbf{Hyb}_2) = 1] - Pr[\mathcal{A}(\mathbf{Hyb}_1) = 1]| > \mathsf{negl}(\lambda)$. The only difference between these two hybrids is that the stored value is 0 and not the secret. Therefore, $\mathcal{A}$ must be able to distinguish between a ciphertext of 0 and an encryption of k under the encryption scheme Π_{pub} or under the encryption scheme Π.

Case: Π_{pub}

First suppose that $\mathcal{A}$ can distinguish between the two hybrids because they can distinguish between an encryption of 0 under Π_{pub} and an encryption of the secret s under Π_{pub}. Then we can construct a reduction $\mathcal{D}_{pub}$ that wins the IND-CPA game against the encryption scheme Π_{pub}. Define $\mathcal{D}_{pub}$ as follows:

$\underline{\mathcal{D}_{pub}(\mathsf{pk}):}$

1. Activate $\mathcal{A}(1^\lambda)$
2. Emulate $\mathbf{Hyb}_1$ for $\mathcal{A}$
3. Upon recovery, query the challenger with $m_0 = (\mathsf{perm\text{-}info}, 0, \perp)$ and $m_1 = (\mathsf{perm\text{-}info}, s, \perp)$ to receive c^*, and use c^* as c_{fin}[11]
4. Output whatever $\mathcal{A}$ outputs

If c^* is an encryption of 0, this is exactly what $\mathcal{A}$ would expect to see at this point in $\mathbf{Hyb}_2$ and if c^* is an encryption of k, this is exactly what $\mathcal{A}$ expects to see in $\mathbf{Hyb}_1$. Therefore $\mathcal{D}_{pub}$ wins the CPA game

[10] Use the backdoor of $\mathcal{G}_{att}$ to ensure the output of $\mathcal{G}_{att}$ is B and a valid signature is received

[11] Use the backdoor of $\mathcal{G}_{att}$ to ensure the output is c^* and a valid signature is received

https://t.me/learningnets

Simulator: $\mathsf{Sim}_{\mathsf{Cloud}^*}$

- **Set Up:** Upon receipt of ($\texttt{register server}, \mathsf{Cloud}^*$) from Cloud^* to $\mathcal{G}_{Perm}$
  1. Send ($\texttt{register server}, \mathsf{Cloud}^*$) to internally simulated $\mathcal{G}_{Perm}$
- **Store:** Upon receipt of ($\texttt{registration request}, \mathsf{Client}, \mathsf{Cloud}^*$) from $\mathcal{G}_{Perm}$
  1. Run internally simulated $\mathcal{G}_{Perm}$ to obtain $\mathsf{perm\text{-}info}$ and send ($\mathsf{perm\text{-}info}, \mathsf{S}$) to Cloud^*
  2. Upon receipt of ($\texttt{storage request}, \mathsf{perm\text{-}info}$) from $\mathcal{F}_{SecRec}$
  3. Set $(\mathsf{vk}_{\mathsf{Client}}, \mathsf{sk}_{\mathsf{Client}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
  4. Sample $a \xleftarrow{\$} \mathbb{Z}_q$ and set $A = g^a$
  5. Send $(A, \mathsf{perm\text{-}info}, \mathsf{vk}_{\mathsf{Client}})$ to Cloud^*
  6. Upon call $\mathcal{G}_{att}.\mathit{install}(\mathsf{perm\text{-}info}, \mathsf{prog}_{SecRec})$ by Cloud^*, run internally simulated $\mathcal{G}_{att}$ and forward response eid to Cloud^*
     (a) If no such call is made, abort with **AttestFail**
  7. Upon call $\mathcal{G}_{att}.\mathit{resume}(\mathsf{eid}, (\text{“store”}, A, \mathsf{vk}_{\mathsf{Client}}))$ by Cloud^*, run internally simulated $\mathcal{G}_{att}$ and forward response $(\mathsf{eid}, \mathsf{prog}_{SecRec}, B, \mathsf{vk}_{\mathsf{Client}}, A, \sigma_{att})$ to Cloud^*
     (a) If no such call is made, abort with **AttestFail**
  8. Upon receipt of $(\mathsf{eid}, \mathsf{prog}_{SecRec}, B, \mathsf{vk}_{\mathsf{Client}}, A, \sigma_{att})$ from Cloud^*, store $\mathsf{retK} \xleftarrow{\$} \mathbb{Z}_q$
  9. Compute $c = \Pi.\mathsf{Enc}(\mathsf{retK}, (\mathsf{perm\text{-}info}, 0, \perp))$ and send c to Cloud^*
  10. Upon call $\mathcal{G}_{att}.\mathit{resume}(\mathsf{eid}, (\text{“verify ciphertext”}, \mathsf{perm\text{-}info}, c))$ by Cloud^*, run internally simulated $\mathcal{G}_{att}$ and use the backdoor to obtain a signature on $b_{att} = 1$ if the correct inputs are supplied and forward response (b_{att}, σ_{att}) to Cloud^*
      (a) If no such call is made, abort with **AttestFail**
- **Remove:** Upon receipt of ($\texttt{removed}, \mathsf{perm\text{-}info}$) from $\mathcal{F}_{SecRec}$
  1. If Cloud^* calls $\mathcal{G}_{att}.\mathit{resume}(\mathsf{eid}, (\text{“remove”}, \mathsf{perm\text{-}info}, \sigma^*_{remove}))$ where σ^*_{remove} was not computed by the simulator, abort with **SigForge**
  2. Compute $\sigma_{remove} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{Client}}, \text{“remove”} \| \mathsf{perm\text{-}info})$
  3. Send ($\texttt{remove}, \mathsf{perm\text{-}info}, \sigma_{remove}$) to Cloud^*
  4. Upon call $\mathcal{G}_{att}.\mathit{resume}(\mathsf{eid}, (\text{“remove”}, \mathsf{perm\text{-}info}, \sigma_{remove}))$ by Cloud^*, run internally simulated $\mathcal{G}_{att}$ to receive output $(\text{“removed”}, \mathsf{perm\text{-}info}, \sigma_{remove}, \sigma_{att})$ and forward the output to Cloud^*
     (a) If no such call is made, abort with **AttestFail**
- **Retrieve:** Upon receipt of ($\texttt{retrieve}, \mathsf{perm\text{-}info}$) from $\mathcal{F}_{SecRec}$
  1. Compute $\sigma_{ret} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{Client}}, \texttt{retrieve} \| \mathsf{perm\text{-}info})$
  2. Send ($\texttt{retrieve}, \mathsf{perm\text{-}info}, \sigma_{ret}$) to Cloud^*
  3. Receive ciphertext c'
  4. If $c' \neq c$ abort with **CTXFail**
  5. Else compute $\Pi.\mathsf{Dec}(\mathsf{retK}, c) \neq (\mathsf{perm\text{-}info}', 0, \mathsf{req} \| \mathsf{res} \| \mathsf{perm})$
  6. If $\mathsf{perm\text{-}info}' \neq \mathsf{perm\text{-}info}$ or $\mathsf{req} \| \mathsf{res} \| \mathsf{perm} \neq \perp$ abort
- **Recover:** Upon receipt of ($\texttt{recover}, \mathsf{perm\text{-}info}$) from $\mathcal{F}_{SecRec}$
  1. Let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Pi_{pub}.\mathsf{Gen}(1^\lambda)$
  2. Let $\mathsf{req} = \text{“}\mathit{recover}\text{”} \| \mathsf{pk}$
  3. Send ($\texttt{generate permission}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}$) to internally simulated $\mathcal{G}_{Perm}$ and receive perm
  4. Send ($\texttt{recover}, \mathsf{perm\text{-}info}, \mathsf{req}, \mathsf{perm}$) to $\mathcal{F}_{SecRec}$
  5. Send perm to Cloud^*
  6. Upon call $\mathcal{G}_{att}.\mathit{resume}(\mathsf{eid}, (\text{“recover”}, \mathsf{pk}, c, \mathsf{req}, \mathsf{perm}))$ by Cloud^*, run internally simulated $\mathcal{G}_{att}$ and forward response $(\mathsf{eid} \| c_{fin}, \sigma_{att})$ where $c_{fin} = \Pi_{pub}.\mathsf{Enc}(\mathsf{pk}, 0)$, and σ_{att} is obtained via backdoor if perm is the correct permissions to Cloud^*
     (a) If no such call is made, abort with **AttestFail**
  7. Receive $(\mathsf{eid} \| c_{fin}, \sigma_{att})$ from Cloud^*

Figure 25: Simulation of Π_{SecRec} for a Malicious Cloud^*

https://t.me/learningnets

with the same non-negligible probability that $\mathcal{A}$ has of distinguishing between the two hybrids, and we have reached our contradiction.

Case: Π

Now suppose that $\mathcal{A}$ is able to distinguish between the two hybrids by submitting a ciphertext $c' \neq c$. That is, $\mathcal{A}$ is able to produce a new ciphertext that decrypts, else $\mathbf{Hyb}_1$ would abort at the same point. Then we can construct a reduction $\mathcal{D}_{CTXT}$ that can win the INT-CTXT game [BN08] with the same non-negligible probability. Define $\mathcal{D}_{CTXT}$ as follows:

$\underline{\mathcal{D}(1^\lambda):}$

1. Activate $\mathcal{A}(1^\lambda)$
2. Emulate $\mathbf{Hyb}_1$ for $\mathcal{A}$
3. Upon storage, query the challenger with $(\mathsf{perm\text{-}info}, s, \perp)$ and receive c
4. Upon retrieval, if $\mathcal{A}$ submits a ciphertext $c' \neq c$, submit c' to the challenger
5. Submit "Finalize" to the challenger

Because $\mathcal{A}$ is able to distinguish between the two hybrids by submitting a ciphertext that is not equal to the ciphertext stored but decrypts, we know that $\mathcal{D}_{CTXT}$ wins the game, also by submitting a ciphertext that was never queried yet decrypts, with the same non-negligible probability that $\mathcal{A}$ has of distinguishing between the two hybrids, and have reached our contradiction.

Suppose instead that $\mathcal{A}$ distinguishes between the two hybrids by distinguishing between an encryption of 0 under Π and encryption of the secret s under Π. Then we can construct a reduction $\mathcal{D}_{CPA}$ that can win the IND-CPA game [BN08] with the same non-negligible probability. Define $\mathcal{D}_{CPA}$ as follows:

$\underline{\mathcal{D}_{CPA}(1^\lambda):}$

1. Activate $\mathcal{A}(1^\lambda)$
2. Emulate $\mathbf{Hyb}_1$ for $\mathcal{A}$
3. Upon storage, query the challenger with $m_0 = (\mathsf{perm\text{-}info}, 0, \perp)$ and $m_1 = (\mathsf{perm\text{-}info}, s, \perp)$ and receive c^*, and use c^* as the ciphertext for storage
4. Output whatever $\mathcal{A}$ outputs

If c^* is an encryption of 0, this is exactly what $\mathcal{A}$ would expect to see at this point in $\mathbf{Hyb}_2$ and if c^* is an encryption of s, this is exactly what $\mathcal{A}$ expects to see in $\mathbf{Hyb}_1$. Therefore $\mathcal{D}_{CPA}$ wins the CPA game with the same non-negligible probability that $\mathcal{A}$ has of distinguishing between the two hybrids, and we have reached our contradiction.

Therefore, in each case, we can construct a reduction that either wins the IND-CPA game against Π_{pub}, the INT-CTXT game against Π, or the CPA game against Π, and have a contradiction. Therefore $\mathbf{Hyb}_2$ is indistinguishable from $\mathbf{Hyb}_1$. □

Lemma 11. *If $\mathcal{G}_{att}$ is parameterized by a EUF-CMA secure signature scheme $\Sigma_{att} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$, $\mathbf{Hyb}_3$ is indistinguishable from $\mathbf{Hyb}_2$*

Proof. Towards a contradiction assume that there exists an adversary $\mathcal{A}$ such that $|Pr[\mathcal{A}(\mathbf{Hyb}_3) = 1] - Pr[\mathcal{A}(\mathbf{Hyb}_2) = 1]| > \mathsf{negl}(\lambda)$. Then we can construct a reduction $\mathcal{D}$ with the goal of winning the unforgeability game against the signature scheme Σ_{att}. Define $\mathcal{D}$ as follows:

$\underline{\mathcal{D}(\mathsf{vk}):}$

1. Activate $\mathcal{A}(1^\lambda)$
2. Emulate $\mathbf{Hyb}_2$ for $\mathcal{A}$
3. Upon submission of a signature σ^* by $\mathcal{A}$ that was not the result of a call to $\mathcal{G}_{att}$, submit σ^* and the message it signs to the challenger

The only difference between the two hybrids is that in $\mathbf{Hyb}_3$ the simulator aborts with **AttestFail** when the adversary does not make the proper calls to $\mathcal{G}_{att}$. Therefore, $\mathcal{A}$ must be able to produce forged signatures that verify, else $\mathbf{Hyb}_2$ would abort at the same point, and we know that $\mathcal{D}$ must then win the unforgeability game with the same non-negligible probability. □

Lemma 12. *If $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$ is an EUF-CMA secure signature scheme, then $\mathbf{Hyb}_4$ is indistinguishable from $\mathbf{Hyb}_3$*

https://t.me/learningnets

Simulator: $\mathsf{Sim}_{\mathsf{Client}^*}$

- **Set Up:** Upon receipt of $(\texttt{registered}, \mathsf{Cloud})$ from $\mathcal{G}_{Perm}$
  1. Send $(\texttt{registered}, \mathsf{Cloud})$ to Client^*
- **Store:** Upon receipt of $(\texttt{register client}, \mathsf{Client}^*, \mathsf{Cloud})$ from Client^* intended for $\mathcal{G}_{Perm}$
  1. Forward $(\texttt{register client}, \mathsf{Client}^*, \mathsf{Cloud})$ to internally simulated $\mathcal{G}_{Perm}$ to obtain $\mathsf{perm\text{-}info}$
  2. Upon receipt of $(A, \mathsf{perm\text{-}info}, \mathsf{vk}_{\mathsf{Client}})$ from Client^* run $\mathcal{G}_{att}.\mathit{install}(\mathsf{perm\text{-}info}, \mathsf{prog}_{SecRec}) = \mathsf{eid}$ on internally simulated $\mathcal{G}_{att}$
  3. Run $\mathcal{G}_{att}.\mathit{resume}(\mathsf{eid}, (\text{"store"}, A, \mathsf{vk}_{\mathsf{Client}})) = (\mathsf{eid}, \mathsf{prog}_{SecRec}, B, \mathsf{vk}_{\mathsf{Client}}, A, \sigma_{att})$ on internally simulated $\mathcal{G}_{att}$ and store retK
  4. Send $(\mathsf{eid}, \mathsf{prog}_{SecRec}, B, \mathsf{vk}_{\mathsf{Client}}, A, \sigma_{att})$ to Client^*
  5. Receive c from Client^* and compute $\Pi.\mathsf{Dec}(\mathsf{retK}, c) = (\mathsf{perm\text{-}info}', s, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$
  6. If $\mathsf{perm\text{-}info}' \neq \mathsf{perm\text{-}info}$ or $\mathsf{req}\|\mathsf{res}\|\mathsf{perm} \neq \perp$ output $\perp$ and abort
  7. Else send $(\texttt{store}, \mathsf{perm\text{-}info}, s)$ to $\mathcal{F}_{SecRec}$ and store $\mathbb{T}_{\mathsf{Sim}}[\mathsf{perm\text{-}info}] = (\mathsf{eid}, c, \perp)$
- **Remove:** Upon receipt of $(\texttt{remove}, \mathsf{perm\text{-}info}, \sigma_{remove})$ from Client^*
  1. If $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{Client}}, \sigma_{remove}, \text{"remove"}\|\mathsf{perm\text{-}info}) \neq 1$ abort
  2. Else send $(\texttt{remove}, \mathsf{perm\text{-}info})$ to $\mathcal{F}_{SecRec}$ and receive $(\texttt{removed}, \mathsf{perm\text{-}info})$
  3. Run internally simulated $\mathcal{G}_{att}.\mathit{resume}(\mathsf{eid}, (\text{"remove"}, \mathsf{perm\text{-}info}, \sigma_{remove})) = (\text{"removed"}, \mathsf{perm\text{-}info}, \sigma_{remove}, \sigma_{att})$ and send the output to Client^*
- **Retrieve:** Upon receipt of $(\texttt{retrieve}, \mathsf{perm\text{-}info}, \sigma_{ret})$ from Client^*
  1. If $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{Client}}, \sigma_{ret}, \texttt{retrieve}\|\mathsf{perm\text{-}info}) \neq 1$ abort
  2. Else send $(\texttt{retrieve}, \mathsf{perm\text{-}info})$ to $\mathcal{F}_{SecRec}$
  3. If $\mathcal{F}_{SecRec}$ sends $(\texttt{recovered}, \mathsf{perm})$ forward $(\texttt{recovered}, \mathsf{perm})$ to Client^*
  4. Else receive (s) from $\mathcal{F}_{SecRec}$ and send c to Client^*
- **Recover:** Upon receipt of $(\texttt{generate permission}, \mathsf{perm\text{-}info}, \mathsf{Cloud}, \mathsf{req})$ from Client^* intended for $\mathcal{G}_{Perm}$
  1. Send $(\texttt{generate permission}, \mathsf{perm\text{-}info}, \mathsf{Cloud}, \mathsf{req})$ to internally simulated $\mathcal{G}_{Perm}$, receive $(\texttt{permission}, \mathsf{perm\text{-}info}, \mathsf{Cloud}, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$, and send to Client^*
  2. Send $(\texttt{recover}, \mathsf{perm\text{-}info}, \mathsf{req}, \mathsf{perm})$ to $\mathcal{F}_{SecRec}$
  3. Upon receipt of $\mathsf{req}'\|\mathsf{res}'\|\mathsf{perm}'$ from Client^*, if $\mathsf{req}' \neq \mathsf{req}$, $\mathsf{res}' \neq \mathsf{res}$, or $\mathsf{perm}' \neq \mathsf{perm}$ abort
  4. Run $\mathcal{G}_{att}.\mathit{resume}(\mathsf{eid}, (\text{"recover"}, \mathsf{pk}, c, \mathsf{req}, \mathsf{perm})) = (\mathsf{eid}\|c_{fin}, \sigma_{att})$ on internally simulated $\mathcal{G}_{att}$
  5. Send $(\mathsf{eid}\|c_{fin}, \sigma_{att})$

Figure 26: Simulation of Π_{SecRec} for a Malicious Client^*

Proof. Towards a contradiction assume that there exists an adversary $\mathcal{A}$ such that $|Pr[\mathcal{A}(\mathbf{Hyb}_3) = 1] - Pr[\mathcal{A}(\mathbf{Hyb}_2) = 1]| > \mathsf{negl}(\lambda)$. Then we can construct a reduction $\mathcal{D}$ such that $\mathcal{D}$ can win the unforgeability game against the signature scheme Σ with the same non-negligible probability. Define $\mathcal{D}$ as follows:

$\underline{\mathcal{D}(\mathsf{vk})}$:

1. Activate $\mathcal{A}(1^\lambda)$
2. Emulate $\mathbf{Hyb}_3$ for $\mathcal{A}$, querying the challenger to compute signatures
3. If $\mathcal{A}$ makes a remove call to $\mathcal{G}_{att}$ using a signature σ^* that is not the result of a query, submit $(\sigma^*, \text{"remove"}\|\mathsf{perm\text{-}info})$ to the challenger

Because $\mathcal{A}$ can distinguish between the two hybrids, and the only difference is that in $\mathbf{Hyb}_4$ the simulator aborts when $\mathcal{A}$ submits a signature that was not computed by the simulator, $\mathcal{A}$ must be able to compute forgeries that verify, else $\mathbf{Hyb}_3$ would abort at the same point. Therefore $\mathcal{D}$ wins the unforgeability game against Σ with the same non-negligible probability that $\mathcal{A}$ has of distinguishing between the two hybrids. □

Case: Malicious Client^* In Figure 26 we present the simulator in the case of a malicious Client^* and prove through a series of hybrids that the view generated by $\mathsf{Sim}_{\mathsf{Client}^*}$ is indistinguishable from the view generated in the real world.

https://t.me/learningnets

Simulator: Sim_{CC^*}

- **Set Up:** Upon receipt of ($\texttt{register server}, \mathsf{Cloud}^*$) from Cloud^* intended for $\mathcal{G}_{Perm}$
  1. Send ($\texttt{register server}, \mathsf{Cloud}^*$) to internally simulated $\mathcal{G}_{Perm}$
- **Store:** Upon receipt of ($\texttt{register client}, \mathsf{Client}^*, \mathsf{Cloud}^*$) from Client^* intended for $\mathcal{G}_{Perm}$
  1. Forward ($\texttt{register client}, \mathsf{Client}^*, \mathsf{Cloud}$) to internally simulated $\mathcal{G}_{Perm}$ to obtain perm-info, and send (perm-info, Cloud^*) to Client^*
  2. Upon call $\mathcal{G}_{att}.install(\text{perm-info}, \mathsf{prog}_{SecRec})$ by Cloud^*, run internally simulated $\mathcal{G}_{att}$ and forward response eid to Cloud^*
  3. Upon call $\mathcal{G}_{att}.resume(\mathsf{eid}, (\text{"store"}, A, \mathsf{vk}_{\mathsf{Client}}))$ by Cloud^*, run internally simulated $\mathcal{G}_{att}$ and forward response $(\mathsf{eid}, \mathsf{prog}_{SecRec}, B, \mathsf{vk}_{\mathsf{Client}}, A, \sigma_{att})$ to Cloud^* and store retK
  4. Upon call $\mathcal{G}_{att}.resume(\mathsf{eid}, (\text{"verify ciphertext"}, \text{perm-info}, c))$ by Cloud^*
     (a) Compute $\Pi.\mathsf{Dec}(\mathsf{retK}, c) = (\text{perm-info}, s, \mathsf{req}\|\mathsf{res}\|\mathsf{perm})$
     (b) Send ($\texttt{store}$, perm-info, s) to $\mathcal{F}_{SecRec}$
     (c) Run internally simulated $\mathcal{G}_{att}$ and forward response (b_{att}, σ_{att}) to Cloud^*
- **Remove:** Upon call $\mathcal{G}_{att}.resume(\mathsf{eid}, (\text{"remove"}, \text{perm-info}, \sigma_{remove}))$ by Cloud^*
  1. If $\Sigma.\mathsf{Vf}(\mathsf{vk}_{\mathsf{Client}}, \sigma_{remove}, \text{"remove"}\|\text{perm-info}) \neq 1$ abort
  2. Send ($\texttt{remove}$, perm-info) to $\mathcal{F}_{SecRec}$
  3. Run internally simulated $\mathcal{G}_{att}$ and forward the output $(\text{"removed"}, \text{perm-info}, \sigma_{remove}, \sigma_{att})$ to Cloud^*
- **Recover:** Upon receipt of ($\texttt{generate permission}$, perm-info, Cloud^*, req) from Client^* intended for $\mathcal{G}_{Perm}$
  1. Send ($\texttt{generate permission}$, perm-info, Cloud^*, req) to internally simulated $\mathcal{G}_{Perm}$, receive ($\texttt{permission}$, perm-info, Cloud^*, $\mathsf{req}\|\mathsf{res}\|\mathsf{perm}$), and send to Client^*
  2. Send ($\texttt{recover}$, perm-info) to $\mathcal{F}_{SecRec}$
  3. Send ($\texttt{recover}$, perm-info, req, perm) to $\mathcal{F}_{SecRec}$
  4. Upon call $\mathcal{G}_{att}.resume(\mathsf{eid}, (\text{"recover"}, \mathsf{pk}, c, \mathsf{req}, \mathsf{perm}))$ by Cloud^* run internally simulated $\mathcal{G}_{att}$ and forward response $(\mathsf{eid}\|c_{fin}, \sigma_{att})$ to Cloud^*

Figure 27: Simulation of Π_{SecRec} for a Malicious Cloud^* and Client^*

Indistinguishability

Lemma 13. *The view generated by* $\mathsf{Sim}_{\mathsf{Client}^*}$ *is indistinguishable from the view generated by the real world adversary controlling a malicious* Client^*

Proof. $\mathsf{Sim}_{\mathsf{Client}^*}$ behaves as an honest Cloud, and needs no special abort cases. Therefore the view generated by $\mathsf{Sim}_{\mathsf{Client}^*}$ is indistinguishable from the view generated by a real world adversary controlling a malicious Client^*. □

Case: Malicious Cloud^* and Client^* In Figure 27 present the simulator in the case where both Cloud^* and Client^* are malicious and prove through a series of hybrids that the view generated by Sim_{CC^*} is indistinguishable from the view generated in the real world.

Proof

Lemma 14. *The view generated by* Sim_{CC^*} *is indistinguishable from the view generated by the real world adversary*

Proof. Sim_{CC^*} honestly simulates $\mathcal{G}_{att}$ towards Cloud^*, and because we know that a simulator exists for $\mathcal{G}_{Perm}$, is able to generate perm-info as expected for Client^*. Therefore the view generated by Sim_{CC^*} is indistinguishable from that of a real world adversary. □

https://t.me/learningnets

Simulator: Sim_H

- **Set Up:** Receive ($\texttt{registered}, \mathsf{Cloud}$) from $\mathcal{G}_{Perm}$
- **Store:** Upon receipt of ($\texttt{registration request}, \mathsf{Client}, \mathsf{Cloud}^*$) from $\mathcal{G}_{Perm}$
  1. Run internally simulated $\mathcal{G}_{Perm}$ to obtain $\mathsf{perm\text{-}info}$ and send $(\mathsf{perm\text{-}info}, \mathsf{S})$ to Cloud^*
  2. Upon receipt of ($\texttt{storage request}, \mathsf{perm\text{-}info}$) from $\mathcal{F}_{SecRec}$
  3. Set $(\mathsf{vk}_{\mathsf{Client}}, \mathsf{sk}_{\mathsf{Client}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
  4. Sample $a \xleftarrow{\$} \mathbb{Z}_q$ and set $A = g^a$
  5. Run internally simulated $\mathcal{G}_{att}$ to receive $\mathsf{eid} = \mathcal{G}_{att}.install(\mathsf{perm\text{-}info}, \mathsf{prog}_{SecRec})$
  6. Run internally simulated $\mathcal{G}_{att}$ to receive $(\mathsf{eid}, \mathsf{prog}_{SecRec}, B, \mathsf{vk}_{\mathsf{Client}}, A\sigma_{att}) = \mathcal{G}_{att}.resume(\mathsf{eid}, (\text{“store”}, A, \mathsf{vk}_{\mathsf{Client}}))$ and store $\mathsf{retK} \xleftarrow{\$} \mathbb{Z}_q$
  7. Compute $c = \Pi.\mathsf{Enc}(\mathsf{retK}, (\mathsf{perm\text{-}info}, 0, \perp))$ and send c to Cloud^*
  8. Run internally simulated $\mathcal{G}_{att}$ to receive $(b_{att}, \sigma_{att}) = \mathcal{G}_{att}.resume(\mathsf{eid}, (\text{“verify ciphertext”}, \mathsf{perm\text{-}info}, c))$ using the backdoor to receive a signature on $b_{att} = 1$
- **Remove:** Upon receipt of ($\texttt{removed}, \mathsf{perm\text{-}info}$) from $\mathcal{F}_{SecRec}$
  1. Compute $\sigma_{remove} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{Client}}, \text{“remove”} \| \mathsf{perm\text{-}info})$
  2. Run internally simulated $\mathcal{G}_{att}$ to receive $(\text{“removed”}, \mathsf{perm\text{-}info}, \sigma_{remove}, \sigma_{att}) = \mathcal{G}_{att}.resume(\mathsf{eid}, (\text{“remove”}, \mathsf{perm\text{-}info}, \sigma_{remove}))$
- **Recover:** Upon receipt of ($\texttt{recover}, \mathsf{perm\text{-}info}$) from $\mathcal{F}_{SecRec}$
  1. Upon receipt of ($\texttt{permission}, \mathsf{perm\text{-}info}, \mathsf{S}, \mathsf{req}\|\mathsf{res}\|\mathsf{perm}$)
  2. Parse $\mathsf{req} = \text{“}recover\text{”}\|\mathsf{pk}$
  3. Send ($\texttt{recover}, \mathsf{perm\text{-}info}, \mathsf{req}, \mathsf{perm}$) to $\mathcal{F}_{SecRec}$
  4. Run internally simulated $\mathcal{G}_{att}$ to receive $(\mathsf{eid}\|c_{fin}, \sigma_{att}) = \mathcal{G}_{att}.resume(\mathsf{eid}, (\text{“recover”}, \mathsf{pk}, c, \mathsf{req}, \mathsf{perm}))$ where c_{fin} is an encryption of 0

Figure 28: Simulation of Π_{SecRec} for Honest Cloud and Client

Case: Honest Cloud and Client Finally, in Figure 28 we present the simulator in the case where all parties are honest and prove through a series of hybrids that the view generated by Sim_H is indistinguishable from the view generated in the real world.

Proof by Hybrids

We prove indistinguishability through a series of hybrids:

- $\mathbf{Hyb}_0$: The real world protocol
- $\mathbf{Hyb}_1$: This is the same as $\mathbf{Hyb}_0$ except that a random key is used as retK instead of the result of the DHKE
- $\mathbf{Hyb}_2$: This is the same as $\mathbf{Hyb}_1$ except that the value stored upon storage is 0 instead of the secret

Lemma 15. *If the DDH assumption holds in the group G of prime order q, $\mathbf{Hyb}_1$ is indistinguishable from $\mathbf{Hyb}_0$*

Proof. Follows from the proof of Lemma 9 □

Lemma 16. *If $\Pi_{pub} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a CPA secure encryption scheme and $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a CCA secure encryption scheme, $\mathbf{Hyb}_2$ is indistinguishable from $\mathbf{Hyb}_1$*

Proof. Follows from the proof of Lemma 10 □

https://t.me/learningnets

References

[ACAA19] Mehmet Aydar, Salih Cemil Cetin, Serkan Ayvaz, and Betül Aygün. Private key encryption and recovery in blockchain. *CoRR*, abs/1907.04156, 2019. pages 8

[BCC+21] Sam Blackshear, Konstantinos Chalkias, Panagiotis Chatzigiannis, Riyaz Faizullabhoy, Irakliy Khaburzaniya, Eleftherios Kokoris-Kogias, Joshua Lind, David Wong, and Tim Zakian. Reactive key-loss protection in blockchains, 2021. pages 6, 7, 14

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996. pages 28

[BG96] M. Bellare and S. Goldwasser. Encapsulated key escrow, 1996. pages 2, 7

[BG97] Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In Richard Graveman, Philippe A. Janson, Clifford Neuman, and Li Gong, editors, *CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, April 1-4, 1997*, pages 78–91. ACM, 1997. pages 2, 7

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000. pages 8, 28

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptol.*, 21(4):469–491, 2008. pages 33

[Bon11] Dan Boneh. Elgamal digital signature scheme. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed*, pages 395–396. Springer, 2011. pages 17, 28

[CHMV17] Ran Canetti, Kyle Hogan, Aanchal Malhotra, and Mayank Varia. A universally composable treatment of network time. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 360–375. IEEE Computer Society, 2017. pages 10

[CZK+19] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 185–200. IEEE, 2019. pages 2

[DCM20] Emma Dauterman, Henry Corrigan-Gibbs, and David Mazières. Safetypin: Encrypted backups with human-memorable secrets. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 1121–1138. USENIX Association, 2020. pages 7

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2002. pages 10

[DSC+15] Tien Tuan Anh Dinh, Prateek Saxena, Ee-Chien Chang, Beng Chin Ooi, and Chunwang Zhang. M2R: enabling stronger privacy in mapreduce computation. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 447–462. USENIX Association, 2015. pages 2

https://t.me/learningnets

[Dwo23] Morris J Dworkin. Advanced encryption standard (aes). 2023. pages 28

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. 196:10–18, 1984. pages 17, 28

[Gan96] Ravi Ganesan. How to use key escrow (introduction to the special section). *Commun. ACM*, 39(3):32–33, 1996. pages 2, 7

[GKL21] Matthew Green, Gabriel Kaptchuk, and Gijs Van Laer. Abuse resistant law enforcement access systems, 2021. pages 7

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. pages 10

[JKKX17] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. TOPPSS: cost-minimal password-protected secret sharing based on threshold OPRF. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings*, volume 10355 of *Lecture Notes in Computer Science*, pages 39–58. Springer, 2017. pages 7

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 456–486. Springer, 2018. pages 7

[KGM19] Gabriel Kaptchuk, Matthew Green, and Ian Miers. Giving state to the stateless: Augmenting trustworthy computation with ledgers, 2019. pages 2, 10, 13, 17

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. pages 8, 9, 10

[KMB15] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 195–206. ACM, 2015. pages 2

[MAB+13] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In Ruby B. Lee and Weidong Shi, editors, *HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, June 23-24, 2013*, page 10. ACM, 2013. pages 2

[Mic92] Silvio Micali. Fair public-key cryptosystems. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 113–138. Springer, 1992. pages 2, 7

[MKE22] Deepak Maram, Mahimna Kelkar, and Ittay Eyal. Interactive authentication. *IACR Cryptol. ePrint Arch.*, page 1682, 2022. pages 7, 14, 17

[MSH+16] Ujan Mukhopadhyay, Anthony Skjellum, Oluwakemi Hambolu, Jon Oakley, Lu Yu, and Richard R. Brooks. A brief survey of cryptocurrency systems. In *14th Annual Conference on Privacy, Security and Trust, PST 2016, Auckland, New Zealand, December 12-14, 2016*, pages 745–752. IEEE, 2016. pages 1

https://t.me/learningnets

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991. pages 17

[PRZB12] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111, 2012. pages 1

[PST17] Rafael Pass, Elaine Shi, and Florian Tramèr. Formal abstractions for attested execution secure processors. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 260–289, 2017. pages 2, 3, 5, 11, 28

[Rip] Ripple. https://ripple.com/. pages 17

[Sca19] Alessandra Scafuro. Break-glass encryption. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, volume 11443 of *Lecture Notes in Computer Science*, pages 34–62. Springer, 2019. pages 2, 3, 6, 7, 8

[Sch] Ricardo Guilherme Schmidt. Secret multisig recovery. https://hackmd.io/@3esmit/rJ4YBV_JI. pages 14

[Sha95] Adi Shamir. Partial key escrow: A new approach to software key escrow. In *Key escrow conference*, 1995. pages 2, 7

[Tora] Tor project anonymity online. https://www.torproject.org/. pages 5

[torb] Torus labs: Open-source key management. https://tor.us/. pages 7

[tre] Recovery process for the trezor model t. https://trezor.io/learn/a/recover-wallet-on-model-t. pages 7

[VFV17] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. Dandelion: Redesigning the bitcoin network for anonymity. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1):22:1–22:34, 2017. pages 5

[ZDB+17] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In Aditya Akella and Jon Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 283–298. USENIX Association, 2017. pages 2

https://t.me/learningnets