AuditPCH: Auditable Payment Channel Hub with Privacy Protection

IEEE TIFS 2024

Yuxian Li, Jian Weng, Junzuo Lai, Yingjiu Li, Jianfei Sun, Jiahe Wu, Ming Li, Pengfei Wu, Robert H. Deng

汇报人：赵路路

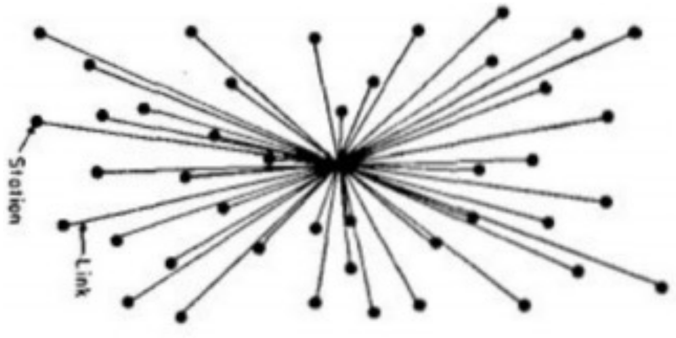
Outline

1. Background
2. Preliminaries
3. Linkable Randomizable Puzzle Scheme
4. Auditable Anonymous PCH
5. Performance Analysis

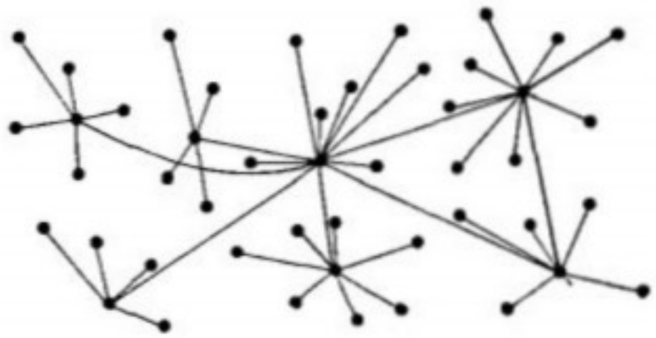
1 Background

1.1 Payment Channel

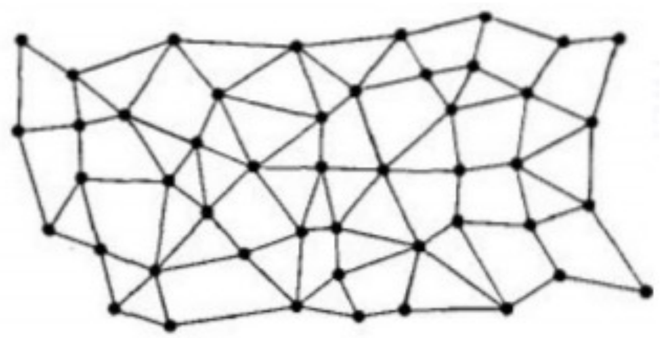
- Two users open a payment channel and perform off-chain payments by updating the channel state enjoying high payment throughput and low confirmation delay.
- If there are more than two users, each pair of users needs to establish their own payment channel to facilitate the payment, which is a non-scalable approach.
- Payment Channel Networks (PCN) enable two users with no direct payment channel to pay each other through the channels of some intermediaries.
- PCN payments may require multi-channel paths and intermediaries to actively participate in relaying the payments, which can lead to their failure.



Centralized



Decentralized



Distributed

1.2 Challenges in Auditable Anonymous PCH

- **Atomicity:** For any payment of m coins from S to R, the PCH should ensure that either R receives m coins from T and T receives m coins from S, or both parties receive none.
- **Value Privacy:** T should not know the payment amount between S and R.
- **Relationship Anonymity:** T should not be able to find out if there is any relation between S and R of a specific payment.
- **Griefing Resistance:** The PCH should only initiate a payment procedure if R can prove that the payment request are previously backed by some coins locked by a S during the payment procedure.
- **Illegal Financial Activity Auditability:** A should know the relationship of S and R and verify the integrity of payments.

1.3 Abstract

- **PCH Definition:**

Anonymous Payment Channel Hub (PCH), one of the most promising layer-two solutions, settles the scalability issue in blockchain while guaranteeing the unlinkability of transacting parties.

- **Problem:**

Developments bring conflicting requirements, i.e., hiding the sender-to-receiver relationships from any third party but opening the relationship to the auditor. Existing works do not support these requirements simultaneously since off-chain transactions are not recorded in the blockchain.

- **This work:**

The first anonymous PCH solution that provides privacy & auditability.

① Linkable randomizable puzzle for conditional transactions.

② A novel auditable solution for PCH.

③ Formal security proof & extensive experiments and evaluation.

2 Preliminaries

2.1 Public Key Encryption

Public Key Encryption. The encryption scheme includes the algorithms (Gen, Enc, Dec) [18]. $(pk, sk) \leftarrow \mathsf{Gen}(\lambda)$ is the key generation algorithm to produce a key pair (sk, pk), where sk is a selected value. $c \leftarrow \mathsf{Enc}(pk, m)$ is the encryption algorithm to encrypt a message m with the public key pk as a ciphertext c. $m' \leftarrow \mathsf{Dec}(c, sk)$ decrypts the ciphertext c as the plaintext m' via the private key sk. We utilize the ElGamal encryption scheme Π_{El} [18] and the Castagnos-Laguillaumie (CL) encryption scheme Π_{CL} which satisfy the Indistinguishability under Chosen Plaintext Attacks (IND-CPA).

SysGen: The system parameter generation algorithm takes as input a security parameter λ. It chooses a cyclic group $(\mathbb{G}, p, g)$ and returns the system parameters $SP = (\mathbb{G}, p, g)$.

KeyGen: The key generation algorithm takes as input the system parameters SP. It randomly chooses $\alpha \in \mathbb{Z}_p$, computes $g_1 = g^\alpha$, and returns a public/secret key pair (pk, sk) as follows:

$$pk = g_1, \quad sk = \alpha.$$

Encrypt: The encryption algorithm takes as input a message $m \in \mathbb{G}$, the public key pk, and the system parameters SP. It chooses a random number $r \in \mathbb{Z}_p$ and returns the ciphertext CT as $CT = (C_1, C_2) = \left(g^r,\ g_1^r \cdot m\right)$.

Decrypt: The decryption algorithm takes as input a ciphertext CT, the secret key sk, and the system parameters SP. Let $CT = (C_1, C_2)$. It decrypts the message by computing

$$C_2 \cdot C_1^{-\alpha} = g_1^r m \cdot (g^r)^{-\alpha} = m.$$

Algorithm $\mathsf{KeyGen}(1^\lambda)$

1. $(B, n, p, s, g, f, G, F) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, 1^\mu)$
2. Pick[a] $x \xleftarrow{\$} \{0, \ldots, Bp-1\}$ and set $h \leftarrow g^x$
3. Set $pk \leftarrow (B, p, g, h, f)$ and $sk \leftarrow x$.
4. Return (pk, sk)

Algorithm $\mathsf{Encrypt}(1^\lambda, pk, m)$

1. Pick $r \xleftarrow{\$} \{0, \ldots, Bp-1\}$
2. Compute $c_1 \leftarrow g^r$
3. Compute $c_2 \leftarrow f^m h^r$
4. Return (c_1, c_2)

Algorithm $\mathsf{Decrypt}(1^\lambda, pk, sk, (c_1, c_2))$

1. Compute $M \leftarrow c_2 / c_1^x$
2. $m \leftarrow \mathsf{Solve}(p, g, f, G, F, M)$
3. Return m

Algorithm $\mathsf{EvalSum}(1^\lambda, pk, (c_1, c_2), (c_1', c_2'))$

1. Compute $c_1'' \leftarrow c_1 c_1'$ and $c_2'' \leftarrow c_2 c_2'$
2. Pick $r \xleftarrow{\$} \{0, \ldots, Bp-1\}$
3. Return $(c_1'' g^r, c_2'' h^r)$

Algorithm $\mathsf{EvalScal}(1^\lambda, pk, (c_1, c_2), \alpha)$

1. Compute $c_1' \leftarrow c_1^\alpha$ and $c_2' \leftarrow c_2^\alpha$
2. Pick $r \xleftarrow{\$} \{0, \ldots, Bp-1\}$
3. Return $(c_1' g^r, c_2' h^r)$

2.2 Commitment Scheme

Commitment scheme. Our construction requires a commitment scheme that allows users to commit a message (e.g., token identification) and verify its correctness. A commitment scheme Π_{com} is composed by three algorithms ($\mathsf{CMSetup}$, Com, $\mathsf{CMVerify}$). $pp \leftarrow \mathsf{CMSetup}(\lambda)$ inputs λ and generates the public parameter pp. $(com, r) \leftarrow \mathsf{Com}(pp, m, r)$ commits the message m in the commitment com with the randomness coin r. $\{0, 1\} \leftarrow \mathsf{CMVerify}(com, r, m)$ verifies if the message m is committed in com. Our solution introduces the Pedersen commitment scheme [14], satisfying the information-theoretically hiding and computationally binding properties.

Let us consider $\mathbb{G} = \langle g \rangle$, a cyclic group of prime order q, and two random generators $g, h \in \mathbb{G}$. The Pedersen commitment scheme allows to commit to scalar elements from $\mathbb{Z}_q$:

Commitment: to commit to a scalar $m \in \mathbb{Z}_q$, one chooses a random $r \xleftarrow{\$} \mathbb{Z}_q$, and sets $c \leftarrow g^m h^r$, while the opening value is set to r;

Opening: to open a commitment $c \in \mathbb{G}$, one reveals the pair (m, r). If $c = g^m h^r$, the receiver accepts the opening to m, otherwise it refuses.

2.3 Malleable Proof Scheme

Malleable proof scheme. The malleable proof schemes [16, 19] support users to transform their receiving proofs into new proofs against the converted witness (e.g., a randomized solution of a randomized puzzle) and statement. Let $R(x, w)$ be a relation related to the language $L := \{x \mid \exists w$ such that $(x, w) \in R\}$, where x is a statement and w is a witness of the statement. Two transformation functions ($w' = \mathcal{T}_{wit}(w)$, $x' = \mathcal{T}_{stmt}(x)$) are defined to restrict the allowed transformation of users. The malleable proof scheme is formulated as: $\Pi_{\mathsf{MP}} = (\mathsf{CRSSetup}, \mathsf{Vry}, \mathsf{Prove}, \mathsf{ZKEval})$. $crs \leftarrow \mathsf{CRSSetup}(\lambda)$ generates the Common Reference Strings (CRS) crs. $\pi \leftarrow \mathsf{Prove}(w, crs, x)$ is the prover algorithm with the witness w, the CRS crs, and the statement x as inputs to produce a proof π stating $(x, w) \in R$. $\{0, 1\} \leftarrow \mathsf{Vry}(\pi, crs, x)$ is the verifier algorithm to check whether the existence of w and x satisfies the relation R. $\pi' \leftarrow \mathsf{ZKEval}(crs, x, \{\mathcal{T}_{wit}, \mathcal{T}_{stmt}\}, \pi)$ produces a transformed proof π' for stating $(x', w') \in R$, where x' and w' come from the transformation functions $\{\mathcal{T}_{wit}, \mathcal{T}_{stmt}\}$. Note that the transformed proof π' still can be verified by the algorithm Vry. Here, a malleable proof scheme [16] is initialized by the Groth-Sahai proof scheme [19] and satisfies the Witness Indistinguishability (WI) property.

Malleable Proof Systems and Applications

Melissa Chase
Microsoft Research Redmond
melissac@microsoft.com

Markulf Kohlweiss
Microsoft Research Cambridge
markulf@microsoft.com

Anna Lysyanskaya
Brown University
anna@cs.brown.edu

Sarah Meiklejohn*
UC San Diego
smeiklej@cs.ucsd.edu

January 16, 2012

Efficient Non-interactive Proof Systems for Bilinear Groups *

Jens Groth
University College London
j.groth@ucl.ac.uk†

Amit Sahai
University of California Los Angeles
sahai@cs.ucla.edu‡

April 7, 2016

2.4 Adaptor Signature

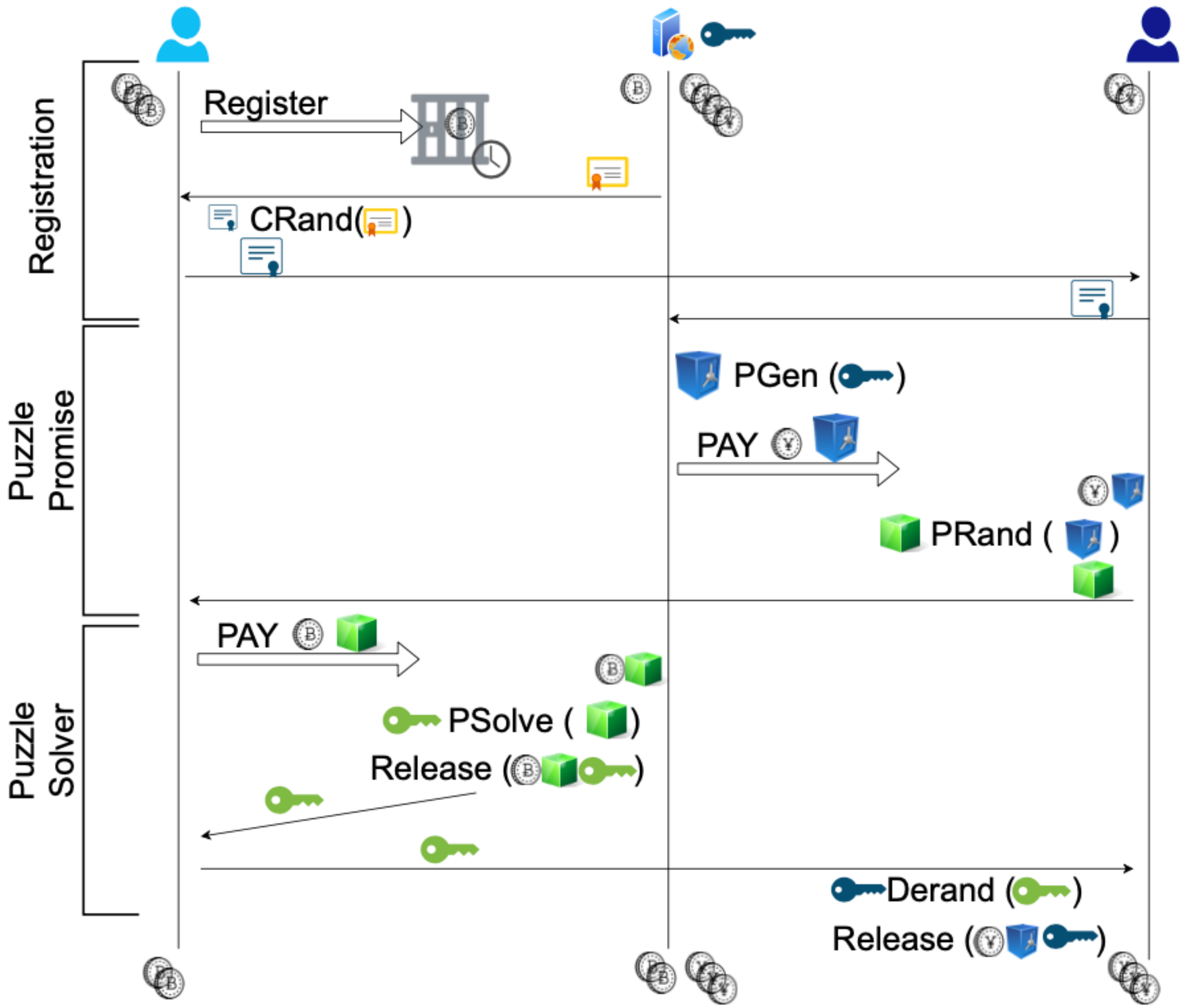
Adaptor signature scheme. Different from traditional signature schemes, the adaptor signature scheme Π_{ad}, which consists of five algorithms (SKeyGen, PSign, Adapt, PVrfy, Ext), enables signers to give a *pre-signature* concerning the revelation of a secret value. We define a statement $Y = g^y$, where g is the generator of the group. $(pk, sk) \leftarrow \mathsf{SKeyGen}(\lambda)$ initializes a key pair (pk, sk) for signing. $\widetilde{\sigma} \leftarrow \mathsf{PSign}(m, Y, sk)$ inputs the secret key sk, a message m, and a statement Y to generate a pre-signature $\widetilde{\sigma}$. $\{0,1\} \leftarrow \mathsf{PVrfy}(pk, m, \widetilde{\sigma}, Y)$ checks the validity of the pre-signature $\widetilde{\sigma}$. $\sigma \leftarrow \mathsf{Adapt}(y, \widetilde{\sigma})$ takes the witness y and the pre-signature $\widetilde{\sigma}$ as inputs to produce a valid signature σ. $y \leftarrow \mathsf{Ext}(\widetilde{\sigma}, \sigma, Y)$ computes the witness y via the inputs $\widetilde{\sigma}$ and σ. The adaptor signature scheme satisfies pre-signature adaptability, which guarantees parties collect a valid signature from a valid pre-signature, and witness extractability, which ensures parties extract a valid witness via a valid signature and its pre-signature. Here, the scheme is formalized in [20, 21] and matches the security property against Existential Unforgeability under Chosen Message Attack (EUF-CMA) [18].

$\mathsf{pSign}_{sk}(m, I_Y)$	$\mathsf{pVrfy}_{pk}(m, I_Y; \tilde{\sigma})$	$\mathsf{Adapt}(\tilde{\sigma}, y)$	$\mathsf{Ext}(\sigma, \tilde{\sigma}, I_Y)$
$x := sk, (Y, \pi_Y) := I_Y$	$X := pk, (Y, \pi_Y) := I_Y$	$(r, \tilde{s}, K, \pi) := \tilde{\sigma}$	$(r, s) := \sigma$
$k \leftarrow_\$ \mathbb{Z}_q, \tilde{K} := g^k$	$(r, \tilde{s}, K, \pi) := \tilde{\sigma}$	$s := \tilde{s} \cdot y^{-1}$	$(\tilde{r}, \tilde{s}, K, \pi) := \tilde{\sigma}$
$K := Y^k, r := f(K)$	$u := \mathcal{H}(m) \cdot \tilde{s}^{-1}$	$\mathbf{return}\ (r, s)$	$y' := s^{-1} \cdot \tilde{s}$
$\tilde{s} := k^{-1}(\mathcal{H}(m) + rx)$	$v := r \cdot \tilde{s}^{-1}$		$\mathbf{if}\ (I_Y, y') \in R'_g$
$\pi \leftarrow \mathsf{P}_Y((\tilde{K}, K), k)$	$K' := g^u X^v$		$\mathbf{then\ return}\ y'$
$\mathbf{return}\ (r, \tilde{s}, K, \pi)$	$\mathbf{return}\ ((I_Y \in L_R) \wedge (r = f(K)) \wedge \mathsf{V}_Y((K', K), \pi))$		$\mathbf{else\ return}\ \perp$

2.5 High-level Design of A^{2}L$^+$

Puzzle-promise phase. During this phase, the receiver P_r starts by sending a valid signature σ_r' on a transaction message m' to the hub P_h. The hub generates a statement/witness pair (A, α) and creates a randomizable puzzle Z along with a zero-knowledge proof π_α [15, 22] that proves α is a valid solution to Z. The hub then produces an adaptor signature $\hat{\sigma}_h'$ over the transaction m' using α and shares both the puzzle and the adaptor signature with P_r. The receiver pre-verifies the signature and randomizes the puzzle Z to Z', which is then shared with the sender P_s, completing the puzzle promise protocol.

Puzzle-solver phase. Here, the sender further randomizes the puzzle Z' to Z'' and generates a pre-signature $\hat{\sigma}_s$ on the transaction m' using Z''. This randomized puzzle and the pre-signature are sent to the hub, which then solves the puzzle Z'' using the trapdoor information to obtain α''. The hub uses α'' to adapt $\hat{\sigma}_s$ into a valid signature σ_s and signs the transaction m' with its secret key, producing σ_h. After verifying the signature σ_s, the hub publishes both σ_s and σ_h. Finally, the secret α'' is extracted and shared with the receiver, allowing them to finalize the transaction by revealing the secret α.



3 Linkable Randomizable Puzzle Scheme

3.1 Randomizable Puzzle Scheme

- Assuming that $(KGen, Enc, Dec)$ is a linearly homomorphic encryption with statistical circuit privacy, the there exists a randomizable puzzle with statistical privacy.

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

$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{PSetup}(1^n)$: is a PPT algorithm that on input security parameter 1^n, outputs public parameters pp and a trapdoor td.

$Z \leftarrow \mathsf{PGen}(\mathsf{pp}, \zeta)$: is a PPT algorithm that on input public parameters pp and a puzzle solution ζ, outputs a puzzle Z.

$\zeta := \mathsf{PSolve}(\mathsf{td}, Z)$: is a *DPT* algorithm that on input a trapdoor td and puzzle Z, outputs a puzzle solution ζ.

$(Z', r) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z)$: is a PPT algorithm that on input public parameters pp and a puzzle Z (which has a solution ζ), outputs a randomization factor r and a randomized puzzle Z' (which has a solution $\phi(\zeta, r)$).

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

$$\mathsf{Enc}(\mathsf{ek}, \zeta) \circ \mathsf{Enc}(\mathsf{ek}, r) = c$$

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

3.2 Linkable Randomizable Puzzle Scheme

The Linkable Randomizable Puzzle (LRP) Scheme

$(pp, sk_0) \leftarrow$ **Setup**(λ): is a PPT algorithm (used by the auditor) that on input security parameter λ, outputs public parameters $pp = (G, g, q, r_0, \Pi_{\mathsf{MP}}, crs, \Pi_{\mathsf{El}}, pk_0)$ and private key sk_0. The details of the outputs are as follows:

- G is an elliptic curve group of order q with generator g.
- r_0 is a random number in Z_q.
- Π_{MP} is a secure malleable non-interactive zero-knowledge proof scheme [16] under a common reference string crs.
- Π_{El} is the ElGamal encryption scheme [18] of which $\mathrm{Enc}(pk_0, \cdot)$ is for encryption and $\mathrm{Dec}(pk_0, sk_0, \cdot)$ is for decryption, and (pk_0, sk_0) is a key pair of Π_{El}.

$(pk_1, sk_1) \leftarrow$ **KGen**(λ) : is a PPT algorithm (used by the hub) that on input security parameter λ, outputs a key pair (pk_1, sk_1) from the Castagnos-Laguillaumie (CL) encryption scheme Π_{CL} [29], of which $\mathrm{Enc}(pk_1, \cdot)$ is for encryption and $\mathrm{Dec}(pk_1, sk_1, \cdot)$ is for decryption.

$(pz, \pi) \leftarrow$ **PGen**$(pp, pk_1, \aleph)$: is a PPT algorithm (used by the hub) that on input public parameters pp, public key pk_1, and a puzzle solution $\aleph \in Z_q$, outputs a puzzle $pz = (\alpha, \beta, \gamma)$ and associated proof π. The details of the outputs are as follows:

- $\alpha = \Pi_{\mathsf{El}} \cdot \mathrm{Enc}(pk_0, r_0)$.
- $\beta = \Pi_{\mathsf{CL}} \cdot \mathrm{Enc}(pk_1, \aleph)$.
- $\gamma = g^{\aleph}$.
- π is a malleable proof of existence of witness $(\aleph, r_0)$ for a statement $\alpha \wedge \beta \wedge \gamma * g^{r_0}$ using Π_{MP}[a].

$0/1 \leftarrow$ **PVerify**(pp, pz, π): is a Deterministic Polynomial Time (DPT) algorithm that on input public parameters pp, a puzzle pz, and a proof π, outputs either 0 (failure) or 1 (success) for verifying pz. The process of generating the outputs is as follows:

- Parse $pz = (\alpha, \beta, \gamma)$.
- Check if proof π is correct for statement $\alpha \wedge \beta \wedge \gamma \wedge \gamma * g^{r_0}$ using scheme Π_{MP}.
- Return 1 if the proof is correct and 0 otherwise.

$\aleph \leftarrow$ **PSol**(pk_1, sk_1, pz): is a DPT algorithm (used by hub) that on input public key pk_1, private key sk_1, and a puzzle pz, outputs $\aleph = \Pi_{\mathsf{CL}}.\mathrm{Dec}(pk_1, sk_1, \beta)$, where pz is parsed as $pz = (\alpha, \beta, \gamma)$.

$(pz', \pi', r) \leftarrow$ **PRand**(pp, pk_1, pz, π): is a PPT algorithm (used by users) that on input public parameters pp, pubic key pk_1, a puzzle pz, and a malleable proof π, outputs a randomized puzzle pz', randomized proof π', and associated random number r. The process of generating the outputs is as follows:

- Parse $pz = (\alpha, \beta, \gamma)$.
- Sample a random number $r \in Z_q$.
- Randomize puzzle pz to $pz' = (\alpha', \beta', \gamma')$ using r such that $\alpha' = \Pi_{\mathsf{El}} \cdot \mathrm{Enc}(pk_0, r_0 + r), \beta' = \Pi_{\mathsf{CL}} \cdot \mathrm{Enc}(pk_1, \aleph + r)$, and $\gamma' = g^{\aleph + r}$ (note that both Π_{CL} and Π_{El} are homomorphic).
- Randomize π to π' using r to prove the existence of witness $(\aleph + r, r_0 + r)$ for statement $\alpha' \wedge \beta' \wedge \gamma' \wedge \gamma' * g^{r_0}$. Note that $T_{wit}((\aleph, r_0)) = (\aleph + r, r_0 + r)$ and $T_{stmt}(\alpha \wedge \beta \wedge \gamma \wedge \gamma * g^{r_0}) = \alpha' \wedge \beta' \wedge \gamma' \wedge \gamma' * g^{r_0}$
- Return $pz' = (\alpha', \beta', \gamma')$, π', and r.

$0/1 \leftarrow$ **PLink**(pp, sk_0, pz, pz'): is a DPT algorithm (used by the auditor) that on input public parameters pp, (auditor's) private key sk_0, and two puzzles pz and pz', outputs 1 (success) or 0 (failure). Respectively, pz and pz' are parsed as $pz = (\alpha, \beta, \gamma)$ and $pz' = (\alpha', \beta', \gamma')$. The process of generating the output is as follows:

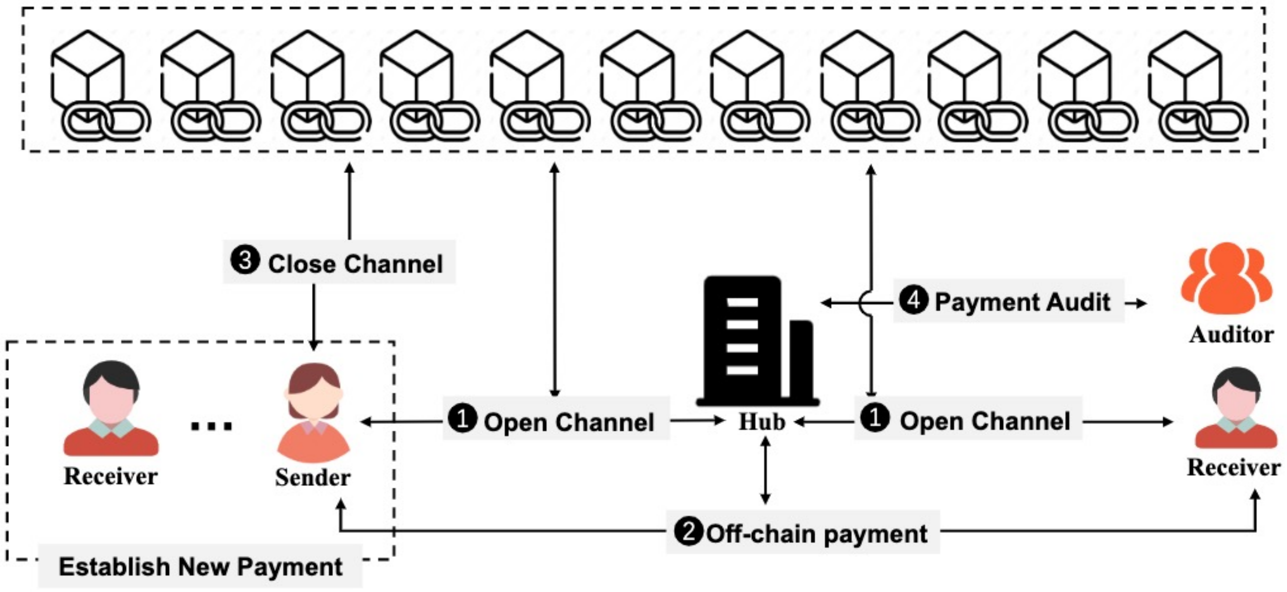
- If $g^{\Pi_{\mathsf{El}}.\mathrm{Dec}(pk_0, sk_0, \alpha')} / g^{\Pi_{\mathsf{El}}.\mathrm{Dec}(pk_0, sk_0, \alpha)} = \gamma' / \gamma$, return 1.
- Otherwise, output 0.

[a] Including $\gamma * g^{r_0}$ in the statement is to bind $\aleph$ to r_0.

4 Auditable Anonymous PCH

4.1 Security Model

- Both hub and users can be malicious while auditor is entrusted with auditing and verifying.
- PPT adversary $\mathcal{A}$ adopts static corruption.
- Synchronous communication network $\mathcal{F}_{syn}$ and secure transmission $\mathcal{F}_{smt}$.
- Blockchain is a global ledger $\mathcal{F}_L$.
- $\mathcal{F}_{AuditPCH}$ defines five operations: setup, open, pay, close, and audit.



4.2 Auditable Anonymous PCH

Sender | **Hub** | **Receiver**

(a) Puzzle generation

Input: $pz', \pi', [\tau_{S\to H}], sk_S, pp, pk_1, pk_0$
Output: $pz'', \pi'', \tilde{\sigma}_{S\to H}, [\tau_{S\to H}]$

1. If **PVerify**(pz', π')=1, continue
2. $(pz'', \pi'', r_2) =$ **PRand**(pp, pk_1, pz, π)
3. $\tilde{\sigma}_{S\to H} =$ **PSign**$([\tau_{S\to H}], \gamma'', sk_S)$

π'': $\{(r_0 + r_1 + r_2, \aleph + r_1 + r_2);$
$\alpha'' = \Pi_{El}.\mathrm{Enc}(pk_0, r_0 + r_1 + r_2) \wedge$
$\beta'' = \Pi_{CL}.\mathrm{Enc}(pk_1, \aleph + r_1 + r_2) \wedge \gamma'' = \gamma' * g^{r_2} \wedge$
$\gamma'' * g^{r_0}\}$

3

Input: $pp, \aleph, [\tau_{H\to R}], sk_H$
Output: $pz, \pi, \tilde{\sigma}_{H\to R}, [\tau_{H\to R}]$

1. $(sk_1, pk_1) =$ **KGen**(λ)
2. $(pz, \pi) =$ **PGen**$(pp, pk_1, \aleph)$, where $(pz = \alpha, \beta, \gamma)$
3. $\tilde{\sigma}_{H\to R} =$ **PSign**$([\tau_{H\to R}], \gamma, sk_H)$

π: $\{(r_0, \aleph);$
$\alpha = \Pi_{El}.\mathrm{Enc}(pk_0, r_0) \wedge$
$\beta = \Pi_{CL}.\mathrm{Enc}(pk_1, \aleph) \wedge$
$\gamma = g^{\aleph} \wedge \gamma * g^{r_0}\}$

Record: $(\sigma_{H\to R}, \tilde{\sigma}_{H\to R}, pz, \pi, [\tau_{H\to R}])$

1

Input: $pz, \pi, \tilde{\sigma}_{H\to R}, [\tau_{H\to R}], pk_1, pp$
Output: π', pz'

1. If **PVrfy**$(\tilde{\sigma}_{H\to R}, [\tau_{H\to R}], pk_H) = 1$, continue
2. If **PVerify**(pz, π)=1, continue
3. $(pz', \pi', r_1) =$ **PRand**(pp, pk_1, pz, π)

π: $\{(r_0 + r_1, \aleph + r_1);$
$\alpha' = \Pi_{El}.\mathrm{Enc}(pk_0, r_0 + r_1) \wedge$
$\beta' = \Pi_{CL}.\mathrm{Enc}(pk_1, \aleph + r_1) \wedge$
$\gamma' = \gamma * g^{r_1} \wedge \gamma' * g^{r_0}\}$

2

(sk_h, pk_h), (sk_S, pk_S) are the key pairs for signing trnasaction

(b) Puzzle solution

Input: $(\sigma_{S\to H}, \tilde{\sigma}_{S\to H}, pz'')$
Output: $\aleph + r_1$

1. $(\aleph + r_1 + r_2) \leftarrow$ **Ext**$(\sigma_{S\to H}, \tilde{\sigma}_{S\to H}, \gamma'')$
2. $\aleph + r_1 = \aleph + r_1 + r_2 - r_2$

5

Input: $pz'', \pi'', \tilde{\sigma}_{S\to H}, [\tau_{S\to H}], sk_1, pk_1$
Output: $\sigma_{S\to H}$

1. $(\aleph + r_1 + r_2) \leftarrow$ **PSol**(pk_1, sk_1, pz'')
2. $\sigma_{S\to H} \leftarrow$ **Adapt**$(\aleph + r_1 + r_2, \tilde{\sigma}_{S\to H})$

Record: $(\sigma_{S\to H}, \tilde{\sigma}_{S\to H}, pz'', \pi'', [\tau_{S\to H}])$

4

Input: $(\aleph + r_1, \tilde{\sigma}_{H\to R})$
Output: $\sigma_{R\to H}$

1. $\aleph = \aleph + r_1 - r_1$
2. $\sigma_{H\to R} \leftarrow$ **Adapt**$(\aleph, \tilde{\sigma}_{H\to R})$

6

(c) Audit

Hub → **Auditor**

Record: $(\sigma_{S\to H}, \tilde{\sigma}_{S\to H}, pz'', \pi'', [\tau_{S\to H}])$

...

Record: $(\sigma_{H\to R}, \tilde{\sigma}_{H\to R}, pz, \pi, [\tau_{H\to R}])$

Input: $(\sigma_{S\to H}, \tilde{\sigma}_{S\to H}, pz'', \pi'', [\tau_{S\to H}]), (\sigma_{H\to R}, \tilde{\sigma}_{H\to R}, pz, \pi, [\tau_{H\to R}]), pp, sk_0$
Output: 1 or 0

1. Check if $\sigma_{S\to H}, \tilde{\sigma}_{S\to H}, \sigma_{H\to R}, \tilde{\sigma}_{H\to R}$ is correct; else output $\perp$
2. $(\aleph + r_1 + r_2) \leftarrow$ **Ext**$(\sigma_{S\to H}, \tilde{\sigma}_{S\to H}, \gamma'')$
3. Check if $\gamma'' = g^{\aleph + r_1 + r_2}$, else output $\perp$
4. $\aleph \leftarrow$ **Ext**$(\sigma_{H\to R}, \tilde{\sigma}_{H\to R}, \gamma)$
5. Check if $1 = \mathrm{PVerify}(pp, pz, \pi)$ and $1 = \mathrm{PVerify}(pp, pz'', \pi'')$, otherwise output $\perp$
6. If **PLink**(pp, sk_0, pz, pz'') =1, output 1; Else output 0.

5 Performance Analysis

5.1 Performance Analysis

- 1.6GHz Intel Core i5-8265U with 8-Core and 8GB RAM.
- Java & C with JPBC and Relic.
- Secp256kl elliptic curve.

TABLE II: The computation cost of the LRP scheme.

	Setup	KGen	PVerify	PGen	PSol	PRand	PLink
Cost(s)	0.419	0.074	6.504	0.615	0.070	0.622	0.013

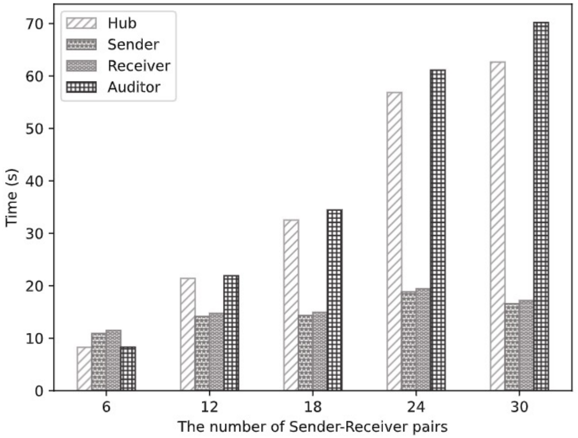


Fig. 4: The computation cost of each role while executing the AuditPCH protocol. The number of Sender-Receiver pairs ranges from 6 to 30.

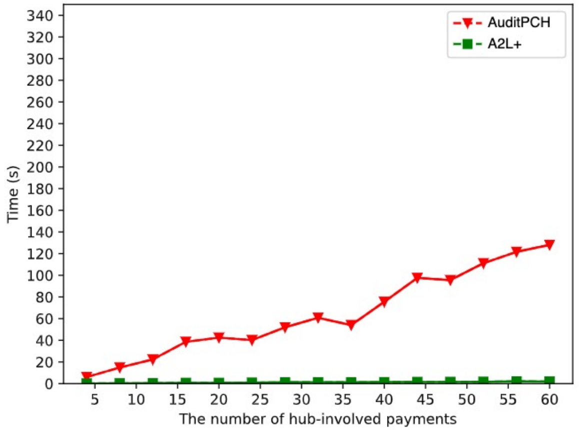


Fig. 5: The time cost comparison between AuditPCH and A^2L^+. The number of hub-involved payments ranges from 4 to 60.

TABLE III: The computation cost of AuditPCH. Time is shown in seconds.

	Pay			Open	Close	Audit	Total
	Channel Authentication	Puzzle Generation	Puzzle Solution				
Sender	0.002	7.279	0.008	–	–	–	7.289
Hub	0.004	0.768	6.582	0.004	–	–	7.358
Receiver	–	7.583	0.008	–	–	–	7.591
Auditor	–	–	–	0.485	–	13.258	13.716

TABLE IV: The communication cost of AuditPCH. n is the number of payments. Size is shown in KB.

	Pay			Open	Close	Audit
	Channel Authentication	Puzzle Generation	Puzzle Solution			
Sender	0.969	5.764	0.281	–	–	–
Hub	0.203	5.764	0.562	–	–	$11.52n + 2n[TX]$
Receiver	–	5.404	–	–	–	–
Auditor	–	–	–	7.230	–	–

Thanks! Questions?