PGC: Decentralized Confidential Payment System with Auditability

Yu Chen	Shandong University
Xuecheng Ma	SKLOIS, CAS
Cong Tang	pgc.info
Man Ho Au	The University of Hong Kong

ESORICS 2020

http://eprint.iacr.org/2019/319

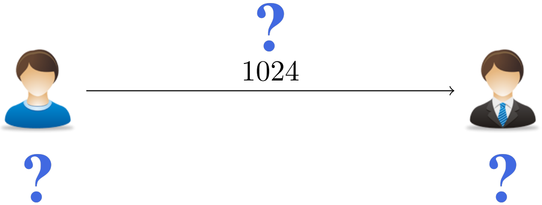
https://github.com/yuchen1024/libPGC

Outline

Outline

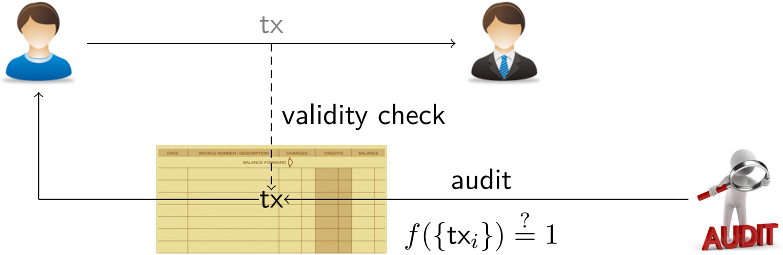
Privacy in Payment System

no one can figure out the transfer amount



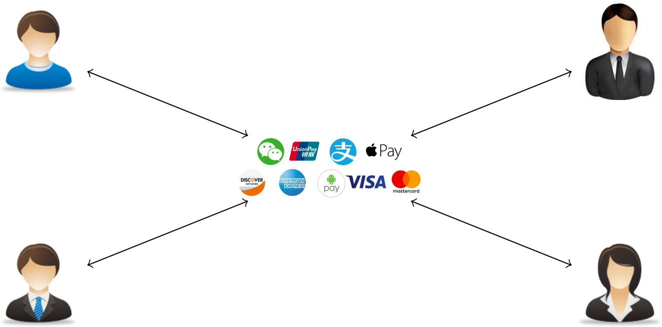
no one can identify the “true” sender and recipient

Auditablity in Payment System



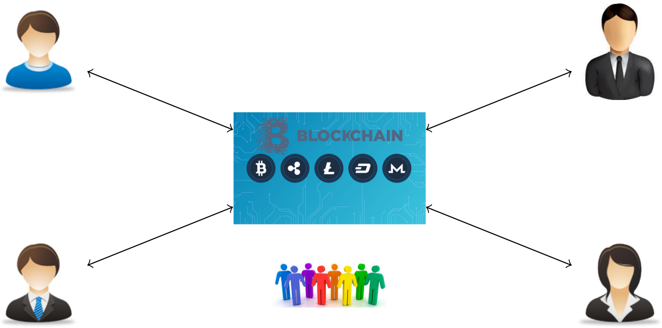
- f denotes the audit predicate that checks if $\{\mathsf{tx}_i\}$ satisfy some specified policy

Centralized Payment System



- txs are kept on a private ledger only known to the center
- the center is in charge of validity check as well as protecting privacy and conducting audit

Decentralized Payment System (Blockchain-based Cryptocurrencies)

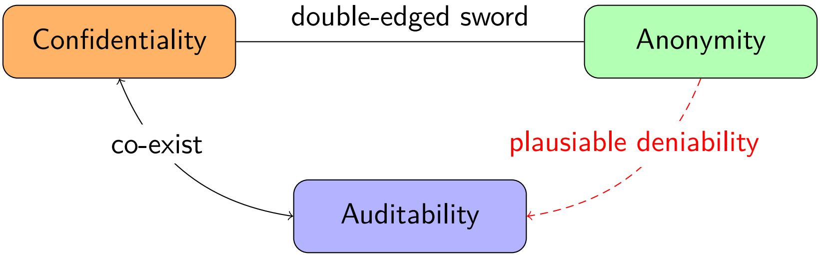


- txs are kept on a global distributed public ledger — the blockchain
- to ensure public verifiability, Bitcoin and Ethereum simply expose all tx information in public $\rightsquigarrow$ no privacy

Motivation

Privacy and Auditability are crucial in any financial system, we want to know:

In the decentralized setting, can we have the good of both?

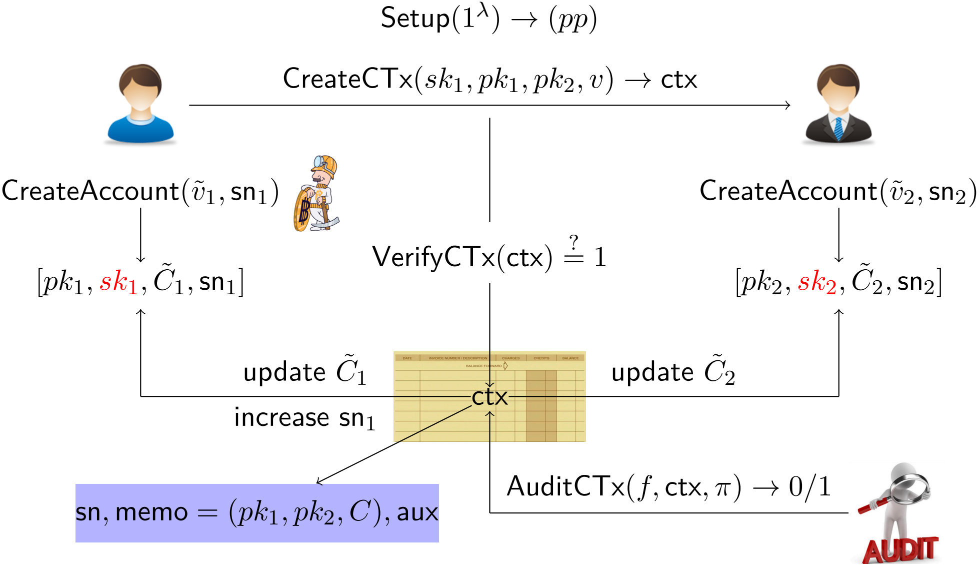


In this work, we trade anonymity for auditablity, propose the first

auditable decentralized confidential payment (DCP) system
in the account-based model

Outline

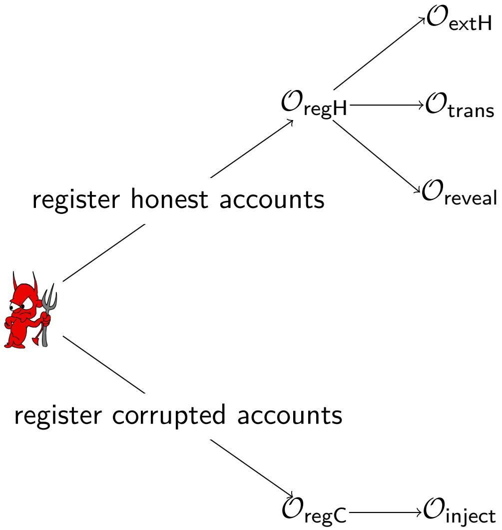
Algorithms of Auditable DCP (Account-based Model)



Desired Functionality and Security

Verifiability		validity of txs are publicly verifiable
Authenticity		only the sender can generate txs, nobody else can forge
Confidentiality		except the sender and receiver, nobody learns the transfer amount
Soundness		even the sender cannot generate an illegal tx that passes validity check
Auditability		particpants cannot cheat and audit is privacy-preserving

Formal Security Model (Oracles)



corrupt honest accounts

direct honest accounts to conduct ctx

ask honest accounts to reveal ctx

inject ctx from corrupted accounts

Formal Security Model: Authenticity

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} \mathsf{VerifyCTx}(\mathsf{ctx}^*) = 1 \ \wedge \\ pk_s^* \in T_{\mathsf{honest}} \wedge \ \mathsf{ctx}^* \notin T_{\mathsf{ctx}}(pk_s^*) \end{array} : \begin{array}{c} pp \leftarrow \mathsf{Setup}(\lambda); \\ \mathsf{ctx}^* \leftarrow \mathcal{A}^{\mathcal{O}}(pp); \end{array}\right].$$

Formal Security Model: Confidentiality

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta = \beta' : \begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ (state, pk_s^*, pk_r^*, v_0, v_1) \leftarrow \mathcal{A}_1^{\mathcal{O}}(pp); \\ \beta \xleftarrow{\mathsf{R}} \{0,1\}; \\ \mathsf{ctx}^* \leftarrow \mathsf{CreateCTx}(sk_s^*, pk_s^*, pk_r^*, v_\beta); \\ \beta' \leftarrow \mathcal{A}_2^{\mathcal{O}}(state, \mathsf{ctx}^*); \end{array}\right] - \frac{1}{2}.$$

To prevent trivial attacks, $\mathcal{A}$ is subject to the following restrictions:

1. pk_s^*, pk_r^* chosen by $\mathcal{A}$ are required to be honest accounts, and $\mathcal{A}$ is not allowed to make corrupt queries to either pk_s^* or pk_r^*;
2. $\mathcal{A}$ is not allowed to make reveal query to ctx^*.
3. let v_{sum} (with initial value 0) be the dynamic sum of the transfer amounts in $\mathcal{O}_{\mathsf{trans}}$ queries related to pk_s^* after ctx^*, both $\tilde{v}_s - v_0 - v_{\mathsf{sum}}$ and $\tilde{v}_s - v_1 - v_{\mathsf{sum}}$ must lie in $\mathcal{V}$.

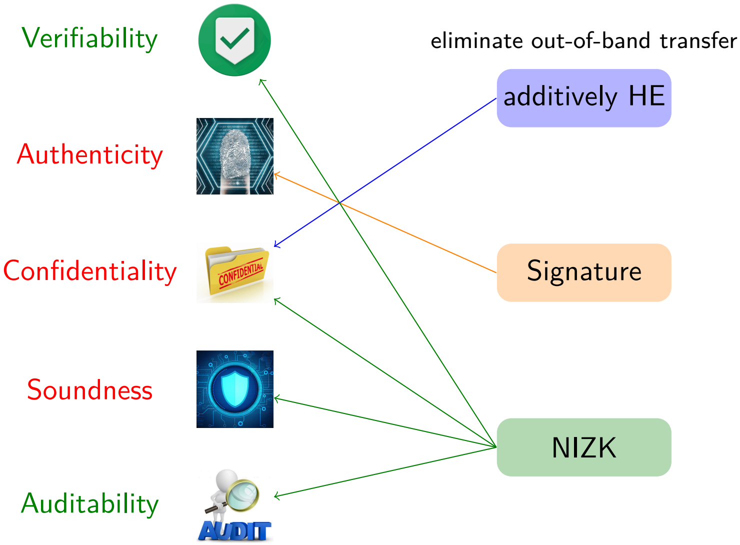
Restrictions 1 and 2 prevents trivial attack by decryption, restrictions 3 prevent inferring β by testing whether overdraft happens.

Formal Security Model: Soundness

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} \mathsf{VerifyCTx}(\mathsf{ctx}^*) = 1 \\ \wedge\ \mathsf{memo}^* \notin L_{\mathsf{valid}} \end{array} : \begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ \mathsf{ctx}^* \leftarrow \mathcal{A}^{\mathcal{O}}(pp); \end{array}\right].$$

Here, $\mathsf{ctx}^* = (\mathsf{sn}^*, \mathsf{memo}^*, \mathsf{aux}^*)$.

Choice of Building Blocks



A Subtle Point: Key reuse vs. Key Separation

We employ PKE and SIG simutaneously to secure auditable DCP.

---

key separation

$(pk_1, sk_1), (pk_2, sk_2)$

Pros

- off-the-shelf & easy to analyze

Cons

- double key size
- tricky address derivation

key reuse

(pk, sk)

Pros

- greatly simplify DCP system
- more efficient

Cons

- case-tailored design

We choose Integrated Signature and Encryption (ISE): one keypair for both encryption and sign, while IND-CPA and EUF-CMA hold in the joint sense

Generic Construction of Auditable DCP: Building blocks

$\mathsf{ISE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Enc}, \mathsf{Dec})$

- PKE component is additively homomorphic over $\mathbb{Z}_p$
- Fix pp, KeyGen naturally induces an $\mathcal{NP}$ relation:

$$\mathsf{R}_{\mathsf{key}} = \{(pk, sk) : \exists r \text{ s.t. } (pk, sk) = \mathsf{KeyGen}(pp; r)\}$$

$\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$

- adaptive soundness
- adaptive ZK

Algorithms of Auditable DCP: 1/4

Setup(1^λ): generate pp for the auditable DCP system

- $pp_{\mathsf{ise}} \leftarrow \mathsf{ISE.Setup}(1^\lambda)$, $pp_{\mathsf{nizk}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$, $crs \leftarrow \mathsf{NIZK.CRSGen}(pp_{\mathsf{nizk}})$
- output $pp = (pp_{\mathsf{ise}}, pp_{\mathsf{nizk}}, crs)$, set $\mathcal{V} = [0, v_{\mathsf{max}}]$

CreateAcct$(\tilde{v}, \mathsf{sn})$: create an account

- $(pk, sk) \leftarrow \mathsf{ISE.KeyGen}(pp_{\mathsf{ise}})$, pk serves as account address
- $\tilde{C} \leftarrow \mathsf{ISE.Enc}(pk, \tilde{v}; r)$

RevealBalance$(sk, \tilde{C})$: reveal the balance of an account

- $\tilde{m} \leftarrow \mathsf{ISE.Dec}(sk, \tilde{C})$

Algorithms of Auditable DCP: 2/4

CreateCTx(sk_s, pk_s, v, pk_r): transfer v coins from account pk_s to account pk_r.

- $C_s \leftarrow \text{ISE.Enc}(pk_s, v; r_1)$, $C_r \leftarrow \text{ISE.Enc}(pk_r, v; r_2)$, memo $= (pk_s, pk_r, C_s, C_r)$.
- run NIZK.Prove with witness (sk_s, r_1, r_2, v) to generate a proof π_{correct} for memo $= (pk_s, pk_r, C_s, C_r) \in L_{\text{valid}} \mapsto L_{\text{equal}} \wedge L_{\text{right}} \wedge L_{\text{solvent}}$

$$L_{\text{equal}} = \{(pk_s, pk_r, C_s, C_r) \mid \exists r_1, r_2, v \text{ s.t. } C_s = \text{ISE.Enc}(pk_s, v; r_1) \wedge C_r = \text{ISE.Enc}(pk_r, v; r_2)\}$$

$$L_{\text{right}} = \{(pk_s, C_s) \mid \exists r_1, v \text{ s.t. } C_s = \text{ISE.Enc}(pk_s, v; r_1) \wedge v \in \mathcal{V}\}$$

$$L_{\text{solvent}} = \{(pk_s, \tilde{C}_s, C_s) \mid \exists sk_1 \text{ s.t. } (pk_s, sk_s) \in \text{R}_{\text{key}} \wedge \text{ISE.Dec}(sk_s, \tilde{C}_s - C_s) \in \mathcal{V}\}$$

- $\sigma \leftarrow \text{ISE.Sign}(sk_s, (\text{sn}, \text{memo}, \pi_{\text{valid}}))$
- output ctx $= (\text{sn}, \text{memo}, \pi_{\text{valid}}, \sigma)$.

Algorithms of Auditable DCP: 3/4

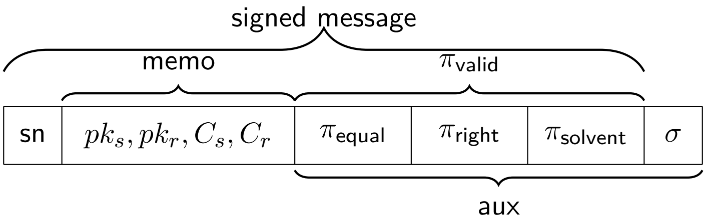


Figure: Data structure of confidential transaction.

VerifyCTx(ctx): check if ctx is valid.

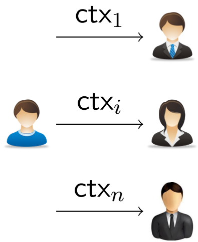
- parse ctx $= (\text{sn}, \text{memo}, \pi_{\text{valid}}, \sigma)$, memo $= (pk_s, pk_r, C_s, C_r)$:
  1. check if sn is a fresh serial number of pk_s (inspect the blockchain);
  2. check if ISE.Verify$(pk_s, (\text{sn}, \text{memo}, \pi_{\text{valid}}), \sigma) = 1$;
  3. check if NIZK.Verify$(crs, \text{memo}, \pi_{\text{valid}}) = 1$.
- ctx is recorded on the ledger if validity test passes or discarded otherwise.

Update(ctx): sender updates his balance $\tilde{C}_s = \tilde{C}_s - C_s$ and increments sn, receiver updates his balance $\tilde{C}_r = \tilde{C}_r + C_r$.

Algorithms of Auditable DCP: 4/4

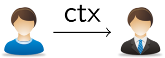
JustifyCTx$(pk, sk, \{\text{ctx}_i\}_{i=1}^n, f)$: user pk runs NIZK.Prove with witness sk to generate a zero-knowledge proof π_f for $f(\{\text{ctx}_i\}_{i=1}^n) = 1$.

$f_{\text{limit}} : \sum_{i=1}^n v_i < \ell$
anti-money laundering



$f_{\text{rate}} : v_1/v_2 = \rho$
tax payment

$f_{\text{open}} : v = v^*$
selective disclosure



AuditCTx$(pk, \{\text{ctx}_i\}_{i=1}^n, f, \pi_f)$: auditor runs NIZK.Verify to check if π_f is valid.

Outline

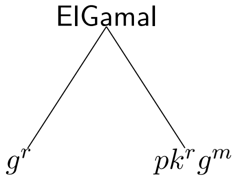
Disciplines in Mind

While the auditbale DCP framework is intuitive, secure and efficient instantiation requires *clever choice and design of building blocks*.

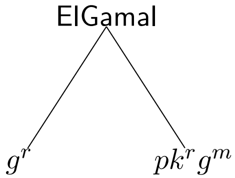
---

efficient	efficient ctx generation/verification compact ctx size
transparent setup	system does not require a trusted setup design case-tailored NIZK
simple & modular	build the system from reusable gadgets can be reused in other places

Encryption Component of ISE



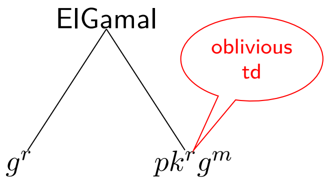
Encryption Component of ISE

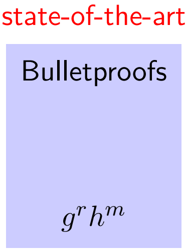


state-of-the-art

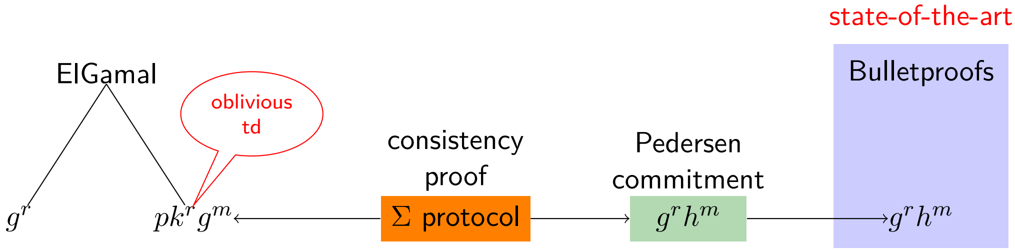
Bulletproofs

Encryption Component of ISE



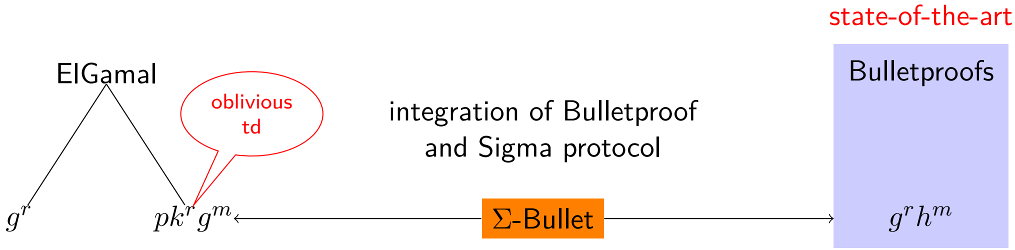


Encryption Component of ISE



Quisquis's approach [FMMO19]
bring extra bridging cost

Encryption Component of ISE



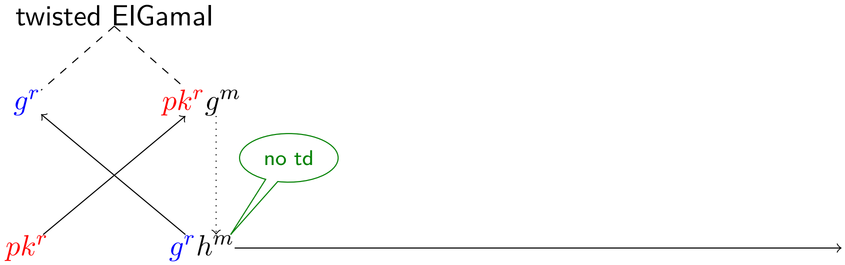
Zether's approach [BAZB20]
require dissecting Bulletproof, not modular

Encryption Component of ISE: Twisted ElGamal

twisted ElGamal

g^r $\quad$ $pk^r g^m$

Encryption Component of ISE: Twisted ElGamal



Encryption Component of ISE: Twisted ElGamal

Encryption Component of ISE: Twisted ElGamal

- encode message over another generator h
- switch key encapsulation and session key
- advantages
  1. as secure and efficient as standard ElGamal;
  2. Bulletproofs-friendly: especially in the aggregated mode

Comparison to ElGamal

	size				efficiency		
ElGamal	pp	pk	sk	C	KeyGen	Enc	Dec
standard	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$\|\mathbb{Z}_p\|$	$\|2\mathbb{G}\|$	1Exp	3Exp+2Add	1Exp+1Add+1DLOG
twisted	$2\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$\|\mathbb{Z}_p\|$	$\|2\mathbb{G}\|$	1Exp	3Exp+2Add	1Exp+1Add+1DLOG

Related works [FMMO19, BAZB20] use brute-force algorithm to decrypt, we use Shanks's algorithm to speed decryption
admits flexible time/space trade-off and parallelization!

Table: Costs of working with Bulletproofs between standard ElGamal and twisted ElGamal: an additional Pedersen commitment and a Sigma protocol for consistency.

ElGamal	size	efficiency
standard	$2\|\mathbb{G}\| + \|\mathbb{Z}_p\|$	4Exp+1Add
twisted	0	0

Comparison to Paillier

Table: Benchmarks of twisted ElGamal and Paillier PKE (32-bit message space and 128-bit security)

timing (ms)	Setup	KeyGen	Enc	Dec	ReRand	Add	Sub	Scalar
Paillier	—	1644.53	32.211	31.367	—	0.0128	—	—
t-ElGamal	21s+6s	0.0151	0.114	1	0.157	0.0031	0.0042	0.093

size (bytes)	public parameters	public key	secret key	ciphertext
Paillier	—	384	384	768
t-ElGamal	66	33	32	66

Signature Component of ISE

We choose Schnorr signature as the signature component.

1. Setup and KeyGen of Schnorr signature are identical to those of twisted ElGamal.
2. Sign of Schnorr signature is irrelevant to Decrypt of twisted ElGamal:
   - Sign(sk, m): pick $r \xleftarrow{R} \mathbb{Z}_p$, set $A = g^r$, compute $e = \mathsf{H}(m, A)$, $z = r + sk \cdot e \bmod p$, output $\sigma = (A, z)$.

Thus we are able to safely implement key reuse strategy to build ISE

- recall Schnorr signature is provably secure by modeling H as RO: simulating signature oracle by programing H without using $sk \Rightarrow$ signatures reveals zero-knowledge of sk

NIZK for L_{equal}

According to our DCP framework and twisted ElGamal, L_{equal} can be written as:

$$\{(pk_1, X_1, Y_1, pk_2, X_2, Y_2) \mid \exists r_1, r_2, v \text{ s.t. } X_i = pk_i^{r_i} \wedge Y_i = g^{r_i} h^v \text{ for } i = 1, 2\}.$$

On statement $(pk_1, pk_2, X_1, X_2, Y_1, Y_2)$, P and V interact as below:

1. P picks $a, b_1, b_2 \xleftarrow{R} \mathbb{Z}_p$, sends $A_1 = pk_1^a$, $A_2 = pk_2^a$, $B_1 = g^a h^{b_1}$, $B_2 = g^a h^{b_2}$ to V.
2. V picks $e \xleftarrow{R} \mathbb{Z}_p$ and sends it to P as the challenge.
3. P computes $z_i = a + er_i$ and $t_i = b_i + ev$ for $i = \{1, 2\}$ using $w = (r_1, r_2, v)$, then sends (z_1, z_2, t_1, t_2) to V. V accepts iff the following four equations hold simultaneously:

$$\begin{aligned} pk_1^{z_1} &= A_1 X_1^e && (1)\\ pk_2^{z_2} &= A_2 X_2^e && (2)\\ g^{z_1} h^{t_1} &= B_1 Y^e && (3)\\ g^{z_2} h^{t_2} &= B_2 Y^e && (4) \end{aligned}$$

NIZK for L_{right}

According to our DCP framework and twisted ElGamal, L_{right} can be written as:

$$\{(pk, X, Y) \mid \exists r, v \text{ s.t. } X = pk^r \wedge Y = g^r h^v \wedge v \in \mathcal{V}\}.$$

For ease of analysis, we additionally define L_{enc} and L_{range} as below:

$$L_{\text{enc}} = \{(pk, X, Y) \mid \exists r, v \text{ s.t. } X = pk^r \wedge Y = g^r h^v\}$$
$$L_{\text{range}} = \{Y \mid \exists r, v \text{ s.t. } Y = g^r h^v \wedge v \in \mathcal{V}\}$$

It is straightforward to verify that $L_{\text{right}} \subset L_{\text{enc}} \wedge L_{\text{range}}$.

- Σ_{enc}: Sigma protocol for L_{enc}
- Λ_{bullet}: Bulletproofs for L_{range}

DL relation between (g, h) is hard $\Rightarrow \Sigma_{\text{enc}} \circ \Lambda_{\text{bullet}}$ is SHVZK PoK for L_{right}

NIZK for L_{solvent}

According to our DCP framework, L_{solvent} can be written as:

$$\{(pk, \tilde{C}, C) \mid \exists sk \text{ s.t. } (pk, sk) \in \mathsf{R}_{\text{key}} \wedge \mathsf{ISE.Dec}(sk, \tilde{C} - C) \in \mathcal{V}\}.$$

$\tilde{C} = (\tilde{X} = pk^{\tilde{r}}, \tilde{Y} = g^{\tilde{r}} h^{\tilde{m}})$ encrypts $\tilde{m}$ of pk under $\tilde{r}$, $C = (X = pk^r, Y = g^r h^v)$ encrypts v under r. Let $C' = (X' = pk^{r'}, Y' = g^{r'} h^{m'}) = \tilde{C} - C$, L_{solvent} can be rewritten as:

$$\{(pk, C') \mid \exists r', m' \text{ s.t. } C' = \mathsf{ISE.Enc}(pk, m'; r') \wedge m' \in \mathcal{V}\}.$$

Prove it as L_{right}? No! r' is unknown.

Solution: refresh-then-prove

1. refresh C' to C^* under fresh randomness r^* $\Leftarrow$ can be done with sk
2. prove $(C', C^*) \in L_{\text{equal}}$ $\Leftarrow$ Sigma protocol Σ_{ddh} (do not need r')
3. prove $C^* \in L_{\text{right}}$

Bonus: two useful proof gadgets

twisted ElGamal + Bulletproofs: prove an encrypted message lies in specific range

- extremely useful in privacy-preserving applications: confidential transaction and secure machine learning

Bonus: two useful proof gadgets

twisted ElGamal + Bulletproofs: prove an encrypted message lies in specific range

- extremely useful in privacy-preserving applications: confidential transaction and secure machine learning

---

pk^r $g^r h^m$

prover is the sender of C
knows both r and m

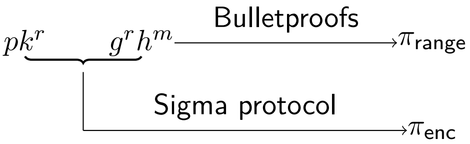
Bonus: two useful proof gadgets

twisted ElGamal + Bulletproofs: prove an encrypted message lies in specific range

- extremely useful in privacy-preserving applications: confidential transaction and secure machine learning

---

prover is the sender of C
knows both r and m



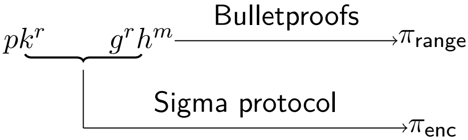
Bonus: two useful proof gadgets

twisted ElGamal + Bulletproofs: prove an encrypted message lies in specific range

- extremely useful in privacy-preserving applications: confidential transaction and secure machine learning

---

prover is the sender of C
knows both r and m



---

prover is the receiver of C
knows sk and thus m

$$pk^{\boxed{r}} \qquad g^{\boxed{r}} h^m$$

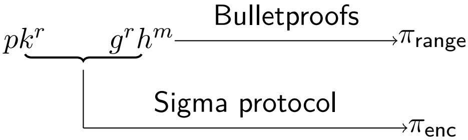
Bonus: two useful proof gadgets

twisted ElGamal + Bulletproofs: prove an encrypted message lies in specific range

- extremely useful in privacy-preserving applications: confidential transaction and secure machine learning

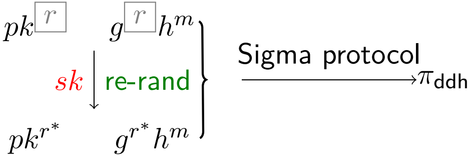
---

prover is the sender of C
knows both r and m



---

prover is the receiver of C
knows sk and thus m

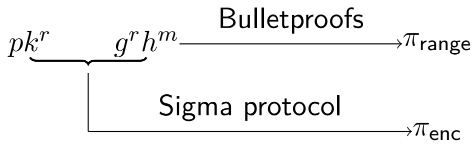


Bonus: two useful proof gadgets

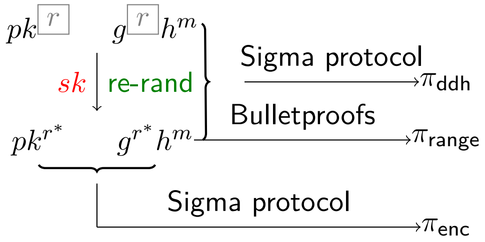
twisted ElGamal + Bulletproofs: prove an encrypted message lies in specific range

- extremely useful in privacy-preserving applications: confidential transaction and secure machine learning

prover is the sender of C knows both r and m



prover is the receiver of C knows sk and thus m



NIZK for Auditing Policies: (1/2)

$$L_{\text{limit}} = \{(pk, \{C_i\}_{1 \le i \le n}, a_{\max}) \mid \exists sk \text{ s.t.}$$
$$(pk, sk) \in \mathsf{R}_{\text{key}} \wedge v_i = \mathsf{ISE.Dec}(sk, C_i) \wedge \sum_{i=1}^{n} v_i \le a_{\max}\}$$

P computes $C = \sum_{i=1}^{n} C_i$, proves $(pk, C) \in L_{\text{solvent}}$ using Gadget-2

$$L_{\text{open}} = \{(pk, C = (X, Y), v) \mid \exists sk \text{ s.t. } X = (Y/h^v)^{sk} \wedge pk = g^{sk}\}$$

$(pk, X, Y, v) \in L_{\text{open}}$ is equivalent to $(Y/h^v, X, g, pk) \in L_{\text{ddh}}$.

NIZK for Auditing Policies: (2/2)

$$L_{\mathsf{rate}} = \{(pk, C_1, C_2, \rho) \mid \exists sk \text{ s.t.}$$
$$(pk, sk) \in \mathsf{R_{key}} \wedge v_i = \mathsf{ISE.Dec}(sk, C_i) \wedge v_1/v_2 = \rho\}$$

We assume $\rho = \alpha/\beta$, where α, β are positive integer much smaller than p.

Let $C_1 = (pk^{r_1}, g^{r_1}h^{v_1})$, $C_2 = (pk^{r_2}, g^{r_2}h^{v_2})$. P computes

$$C_1' = \beta \cdot C_1 = (X_1' = pk^{\beta r_1}, Y_1' = g^{\beta r_1}h^{\beta v_1})$$
$$C_2' = \alpha \cdot C_2 = (X_2' = pk^{\alpha r_2}, Y_2' = g^{\alpha r_2}h^{\alpha v_2})$$

Note $v_1/v_2 = \rho = \alpha/\beta$ iff $h^{\beta v_1} = h^{\alpha v_2}$. $(pk, C_1, C_2, \rho) \in L_{\mathsf{rate}}$ is equivalent to $(Y_1'/Y_2', X_1'/X_2', g, pk) \in L_{\mathsf{ddh}}$.

Due to nice algebra structure of twisted ElGamal, PGC supports efficient audit for any policy that can be expressed as linear constraint over transfer amount and balance

Optimizations

sn	pk_s, pk_r, C_s, C_r	$\pi_{\mathsf{equal}} \circ (\pi^1_{\mathsf{enc}} \circ \pi^1_{\mathsf{bullet}}) \circ (\tilde{C}_s \circ \pi_{\mathsf{ddh}} \circ \pi^2_{\mathsf{enc}} \circ \pi^2_{\mathsf{bullet}})$	σ
	memo		

Optimizations

randomness reuse ↓

sn	pk_s, pk_r, C_s, C_r	$\pi_{\text{equal}} \circ (\pi^1_{\text{enc}} \circ \pi^1_{\text{bullet}}) \circ (\tilde{C}_s \circ \pi_{\text{ddh}} \circ \pi^2_{\text{enc}} \circ \pi^2_{\text{bullet}})$	σ

memo (under pk_s, pk_r, C_s, C_r)

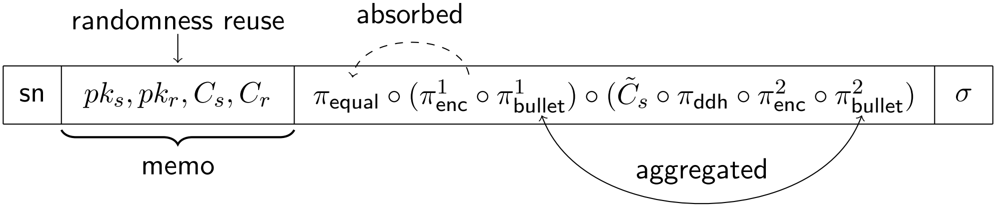
Randomness-Reusing

- original construction encrypts the same message v under pk_1 and pk_2 using independent random coins: $(pk_s, pk_s^{r_1}, g^{r_1}h^v, pk_r, pk_r^{r_2}, g^{r_2}h^v)$
- twisted ElGamal is IND-CPA secure in 1-message/2-recipient setting

$$\text{safe to reuse randomness} \Rightarrow (pk_1, pk_1^r, pk_2, pk_2^r, g^r h^v)$$

Benefit: compact ctx size & simpler design of Σ_{enc}

Optimizations

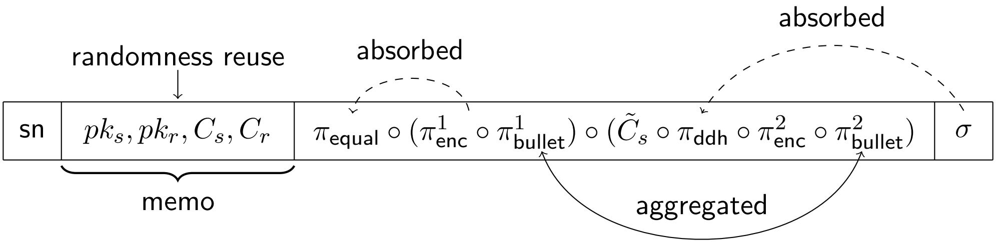


More Efficient Assembly of NIZK

- π_{enc} can be removed since π_{equal} already proves knowledge of C_s
- nice feature of twisted ElGamal $\Rightarrow$ two Bulletproofs can be generated and verified in aggregated mode $\rightsquigarrow$ reduce the size of range proof part by half

Benefit: further shrink the ctx size

Optimizations



Eliminate Explicit Signature

- Σ_{ddh} (3-move public-coin ZKPoK of sk_1) is a sub-protocol of NIZK for L_{solvent}
- apply FS transform by appending the rest part to hash input $\rightsquigarrow$ π_{ddh} serves as both a proof of DDH tuple and a sEUF-CMA signature of ctx (jointly secure with twisted ElGamal)

Benefit: further shrink the ctx size & speed ctx generation/verification

Performance

Table: The computation and communication complexity of PGC.

PGC	ctx size		transaction cost (ms)	
	big-$\mathcal{O}$	bytes	generation	verify
transaction	$(2\log_2(\ell)+20)\lvert\mathbb{G}\rvert+10\lvert\mathbb{Z}_p\rvert$	1310	40	14
auditing	proof size		auditing cost (ms)	
	big-$\mathcal{O}$	bytes	generation	verify
limit policy	$(2\log_2(\ell)+4)\lvert\mathbb{G}\rvert+5\lvert\mathbb{Z}_p\rvert$	622	21.5	7.5
rate policy	$2\lvert\mathbb{G}\rvert+1\lvert\mathbb{Z}_p\rvert$	98	0.55	0.69
open policy	$2\lvert\mathbb{G}\rvert+1\lvert\mathbb{Z}_p\rvert$	98	0.26	0.42

- We set the maximum number of coins as $v_{\max} = 2^{\ell} - 1$, where $\ell = 32$.
- Choose EC curve prime256v1 (128 bit security), $|\mathbb{G}| = 33$ bytes, $|\mathbb{Z}_p| = 32$ bytes.

Comparison to Related Works

Table: Comparison to other account-based DCP

Scheme	transparent setup	scalability	confidentiality	anonymity	auditability
zkLedger	$\checkmark +$ DL	$O(n)$	?	$\checkmark$	$O(m, \lvert f \rvert)$
Zether	$\checkmark +$ DL	$O(1)$	$\checkmark$	$\checkmark$	?
PGC	$\checkmark +$ DL	$O(1)$	$\checkmark$	✗	$O(\lvert f \rvert)$

- n is the number of system users, m is the number of all transactions on the ledger
- zkLedger [NVV18]: (i) ctx size is linear of n, and n is fixed at the very beginning. (ii) confidentiality is questionable due to the use of correlated randomness; (iii) audit efficiency is linear of both m and $|f|$ due to anonymity
- Zether [BAZB20]: (i) possibly support audit when sacrificing anonymity; (ii) security of ZKP is hard to check

Outline

Summary

We propose a framework of auditable DCP from ISE and NIZK

- with formal security model and rigorous proof

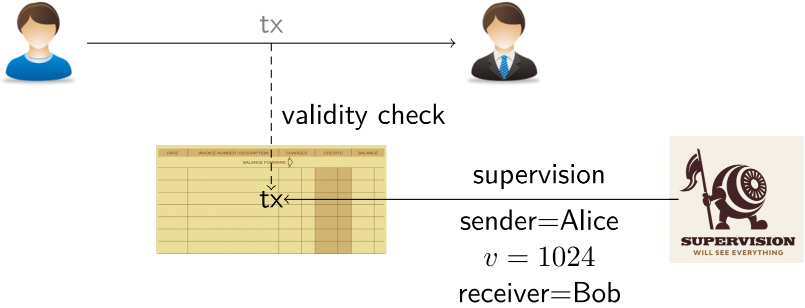
We instantiate the auditable DCP by carefully designing and combining cryptographic primitives $\rightsquigarrow$ PGC

- transparent setup, security solely based on the DLOG assumption
- modular, simple and efficient
- efficient and fine-grained audit

Highlights

- twisted ElGamal: efficient, homomorphic and zero-knowledge proof friendly $\rightsquigarrow$ a good alternative to ISO standard HE schemes: ElGamal and Paillier
- two proof gadgets: widely applicable in privacy-preserving scenarios, e.g. secure machine learning

Ongoing work: Supervisiable DCP



Thanks for Your Attention!

Any Questions?

Reference I

[BAZB20] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *Financial Cryptography and Data Security - FC 2020*, 2020.

[FMMO19] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In *Advances in Cryptology - ASIACRYPT 2019*, volume 11921 of *Lecture Notes in Computer Science*, pages 649–678. Springer, 2019.

[NVV18] Neha Narula, Willy Vasquez, and Madars Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018*, pages 65–80, 2018.