Privacy-Preserving Account-Based Cryptocurrency

Yu Chen
Shandong University

Tutorial based on the following joint work

- **Yu Chen**, Xuecheng Ma, Cong Tang, Man Ho Au
  PGC: Decentralized Confidential Payment System with Auditability
  ESORICS 2020

Outline

Outline

1. Background
2. Framework of PPABC
   - Syntax and Definition
   - Formal Security Model
   - Generic Construction
3. An Efficient Instantiation: PGC
4. Experimental Results
5. Summary

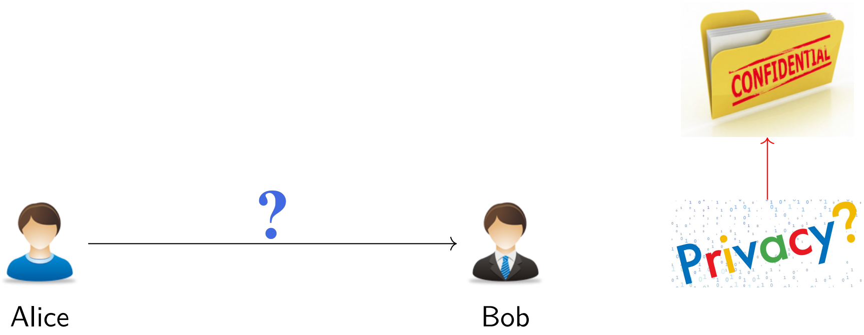
Privacy in Payment System

Alice

1024

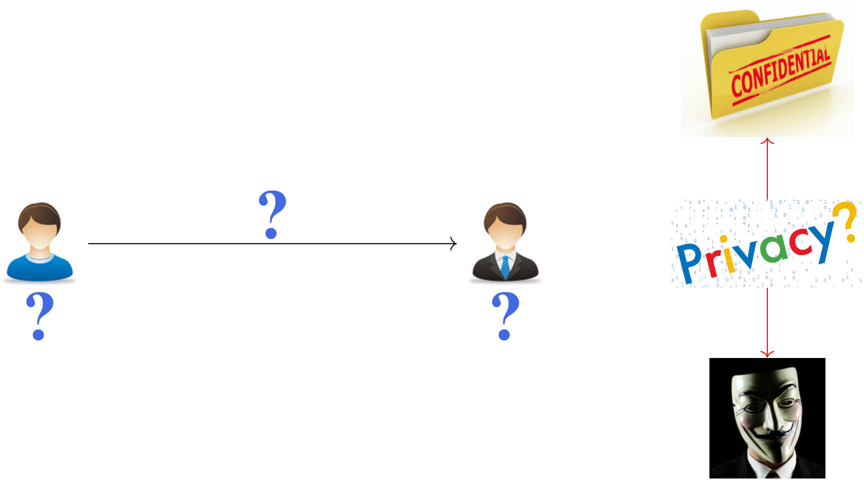
Bob

Privacy in Payment System



Confidentiality: transfer amount is hidden from an external observer

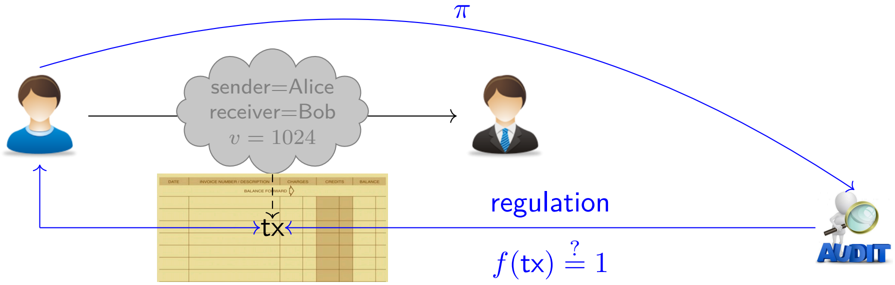
Privacy in Payment System



Confidentiality: transfer amount is hidden from an external observer

Anonymity: identities of sender and receiver is hidden from an external observer

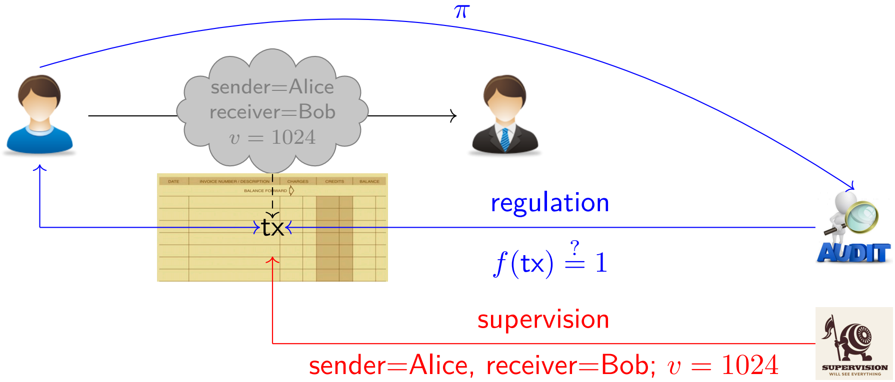
Auditing in Payment System



Regulation: Auditor can verify if txs comply with policies by inquiring users

- auditor does not own extra privilege $\rightsquigarrow$ auditing is interactive

Auditing in Payment System



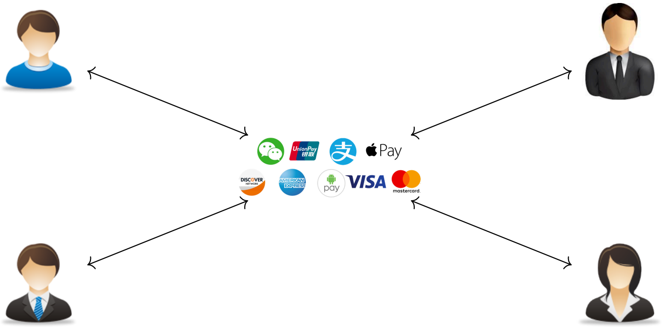
Regulation: Auditor can verify if txs comply with policies by inquiring users

- auditor does not own extra privilege $\rightsquigarrow$ auditing is interactive

Supervision: Auditor can inspect txs of individual user or global users

- auditor owns extra privilege $\rightsquigarrow$ auditing is non-interactive

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 (UTXO model) and Ethereum (account-based model) simply expose all tx information in public $\rightsquigarrow$ no privacy

UTXO vs Account-Based Model

Table: UTXO Model vs. Account-Based Model Comparison

Aspect	UTXO Model	Account-Based Model
Concept	track unspent outputs (like cash)	track account balance (like bank)
Privacy	new add. per tx enhance privacy	add. reuse reduce privacy
Scalability	parallel validation	sequential state updates
Functionality	simple value transfers	Turing-complete smart contracts
Complexity	manage multiple UTXOs	manage a single account

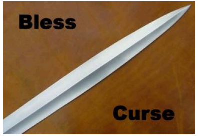
Account-based excels for DeFi/dApps and simpler for users, while it is more challenging to attain privacy. In this work, we focus on account-based cryptocurrencies.

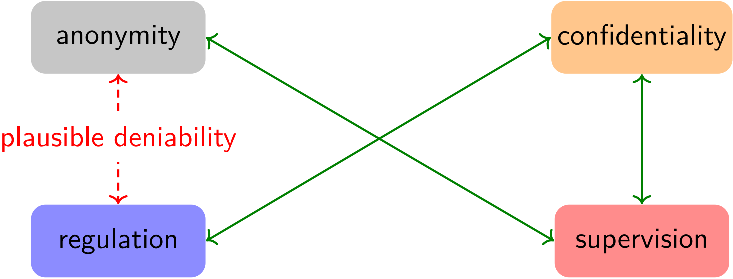
Motivation

Privacy and Auditability are crucial in any financial system. We want to know:

How to achieve both in the decentralized setting?

strong privacy





Motivation

In this work, we trade anonymity for regulation:

~~anonymity~~ confidentiality

regulation supervision

propose Privacy-Preserving Account-Based Cryptocurrency (PPABC) that offers confidentiality and supports regulation + supervision

Outline

Outline

Data Structures of PPABC: 1/2

Blockchain. PPABC operates on top of a publicly accessible and append-only ledger (blockchain) B.

Public parameters. A trusted party generates public parameters pp at the setup time, which is used by system's algorithms.

- pp includes an integer v_{max} that specifies the maximum possible number of coins in the system. Any balance and transfer must lie in $\mathcal{V} = [0, v_{\mathsf{max}}]$.

Account. Each account is associated with a keypair (pk, sk) and an encoded balance $\tilde{C}$ (which encodes plaintext balance $\tilde{v}$).

- both pk and $\tilde{C}$ are public.
- pk serves as account address, which is used to receive transactions from other accounts.
- sk is kept privately, which is used to direct transactions to other accounts and decodes encoded balance.

Data Structures of PPABC: 2/2

Confidential transaction. ctx consists of two parts, memo and aux.

- memo $= (pk_s, pk_r, C)$ records basic information of a transaction from pk_s to pk_r, where C is the encoding of transfer amount,
- aux denotes the auxiliary information, which is application-dependent.

Policies. Let $\{\mathsf{ctx}_i\}_{i=0}^n$ be ctxs related to pk, and v_i be the transfer amount of ctx_i. Policies over $\{v_i\}_{i=1}^n$ are satisfied iff $f(pk, \{\mathsf{ctx}_i\}_{i=0}^n) = 1$, where f is the associated predicate.

- The basic legality policy $f_{\mathsf{legal}}(pk, \mathsf{ctx})$ requires the transfer amount lies in the correct range and the sender account is solvent

We list more application-dependent regulation policies as below:

- limit policy ——$\sum_i^n v_i \leq a_{\max}$: $f_{\mathsf{limit}}(pk, \{\mathsf{ctx}_i\}_{i=1}^n)$
- rate policy ——$v_1/v_2 = \rho$: $f_{\mathsf{rate}}(pk, (\mathsf{ctx}_1, \mathsf{ctx}_2))$
- open policy ——$v = v^*$: $f_{\mathsf{open}}(pk, \mathsf{ctx})$

Entities of PPABC

In PPABC, there are the following types of entities:

- **Users:** each user may control several accounts.
- **Validator:** checking the validity of proposed transactions.
- **Regulator:** checking if a given set of transactions satisfies regulation policies by inquiring involved users.
  - regulator mirrors authorities in the real world, and do not hold any secret
- **Supervisor:** inspecting any transaction without interaction with involved users.
  - supervisor owns some secret

Syntax of PPABC: 1/3

Setup(1^λ): output public parameter pp and possibly an associated secret parameter sp

- A trusted party executes this algorithm once-for-all to setup the whole system. pp will be used as an implicit input in the rest algorithms.

CreateAccount$(\tilde{v})$: on input an initial balance $\tilde{v}$, output a keypair (pk, sk) and an encoded balance $\tilde{C}$.

- A user runs this algorithm to create an account.

RevealBalance$(sk, \tilde{C})$: on input a secret key sk and an encoded balance $\tilde{C}$, output the balance $\tilde{v}$ in plaintext.

- A user runs this algorithm to reveal the balance.

Syntax of PPABC: 2/3

CreateCTx(sk_s, pk_s, pk_r, v): on input a keypair (sk_s, pk_s) of sender account, a receiver account address pk_r, and a transfer amount v, output a confidential transaction ctx.

- A user runs this algorithm to transfer v coins from account pk_s to account pk_r.

VerifyCTx(ctx): on input a ctx, output “0” denotes valid and “1” denotes invalid.

- Validators run this algorithm to check the validity of purported ctx. If ctx is valid, it will be recorded on the blockchain B. Otherwise, it is discarded.

UpdateCTx(ctx): for each fresh ctx appearing on the blockchain B, the corresponding sender and receiver update their encoded balances to reflect the change

- Sender account decreases with v coins while the receiver account increases with v coins.

Syntax of PPABC: 3/3

JustifyCTx$(pk, sk, \{\text{ctx}\}, f)$: on input a user's keypair (pk, sk), a set of ctxs pk involved and a policy f, output a proof π for $f(pk, \{\text{ctx}\}) = 1$.

- A user runs this algorithm to generate a proof for auditing.

AuditCTx$(pk, \{\text{ctx}\}, f, \pi)$: on input a user's public key, a set of ctxs pk involved, a policy f and a proof π, output "0" denotes accept and "1" denotes reject.

- A regulator runs this algorithm to check if $f(pk, \{\text{ctx}\}) = 1$.

OpenCTx(sp, ctx): on input secret parameter sp, output the transaction amount of ctx.

- A supervisor runs this algorithm to inspect ctxs.

Desired Feature and Security

Verifiability	validity of txs are publicly verifiable
Authenticity	only owner can generate tx; nobody else can forge
Confidentiality	external observer does not learn the transfer amount
Soundness	nobody cannot generate an illegal tx that passes validity check
Regulation	user cannot cheat and regulation does not leak more info other than auditing result
Supervision	auditor can see everything, but unable to compromise authenticity

Security Model of PPABC

Formalizing security model for PPABC turns out to be tricky

- strong enough to capture all possible real-world attacks
- clean and handy to use

Outline

Formal Security Model (Oracles)

register honest accounts → $\mathcal{O}_{\text{regH}}$

- $\mathcal{O}_{\text{regH}}$ → $\mathcal{O}_{\text{extH}}$: corrupt honest accounts
- $\mathcal{O}_{\text{regH}}$ → $\mathcal{O}_{\text{trans}}$: direct honest accounts to conduct ctx
- $\mathcal{O}_{\text{regH}}$ → $\mathcal{O}_{\text{reveal}}$: ask honest accounts to reveal ctx

register corrupted accounts → $\mathcal{O}_{\text{regC}}$

- $\mathcal{O}_{\text{regC}}$ → $\mathcal{O}_{\text{inject}}$: inject ctx from corrupted accounts

Formal Security Model: Authenticity

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

- ctx^* is a confidential transaction from target account pk_s^*,
- $T_{\mathsf{ctx}}(pk_s^*)$ denotes the set of all the ctxs originated from pk_s^* in T_{ctx}.

Authenticity require unforgeability w.r.t. entire $\mathsf{ctx}^* = (\mathsf{memo}^*, \mathsf{aux}^*) \rightsquigarrow$ rather strong

- unauthorized transfers from pk_s likely diverge from sender's original intention only when the adversary (without the knowledge sk_s) manages to craft a valid ctx with different memo, because it encodes the core information of a transaction.

Weak authenticity: only requiring unforgeability against memo^*
$\Rightarrow$ allow us to eliminate explicit signature

Formal Security Model: Confidentiality

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

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

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

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

- authenticity and confidentiality are defined w.r.t. outsider adversaries (without secret key)
- soundness is defined w.r.t. both outsider and insider adversaries (even with secret key)

Formal Security Model: Secure Auditing

For regulation compliance, we require:

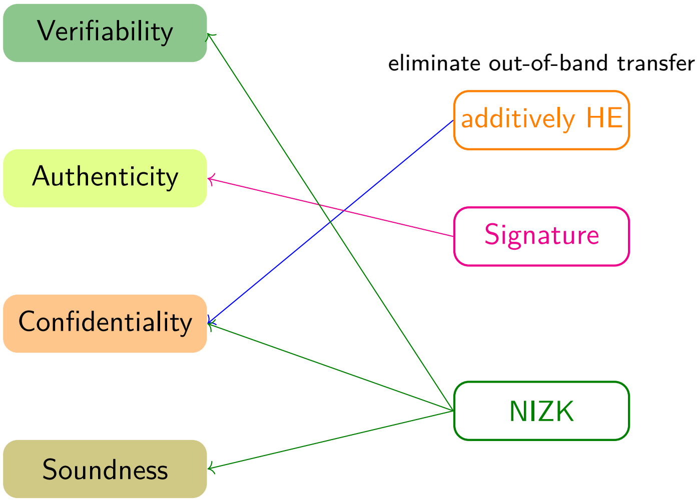
- **Correctness**: no PPT adversary can fool the regulator to accept a false auditing result.
- **Minimal information disclosure**: the regulator learns nothing other than the auditing result.

For supervision, we require:

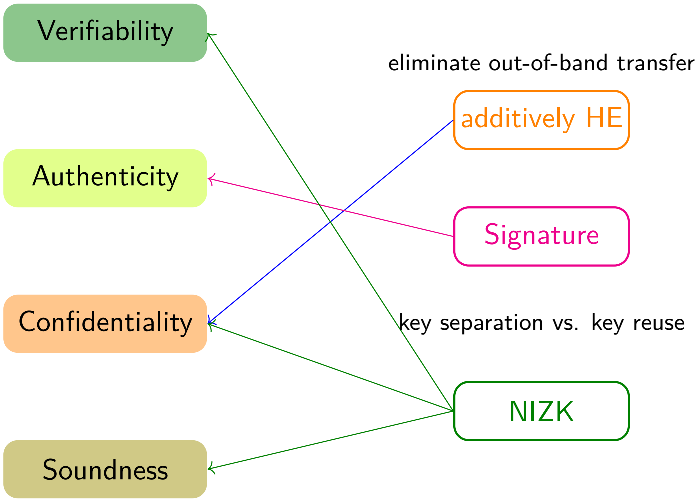
- **Consistency**: no PPT adversary can generate a transaction such that supervisor's view is different from the real receiver's view.
- **Safefy**: even the supervisor with sp cannot break the authenticity and soundness.

Outline

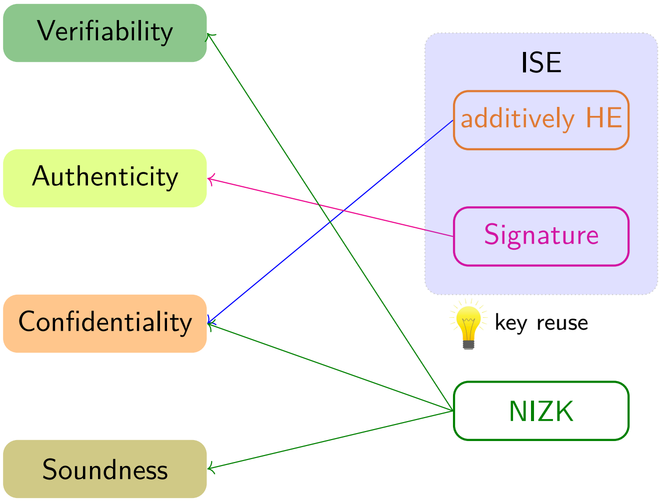
Choice of Building Blocks



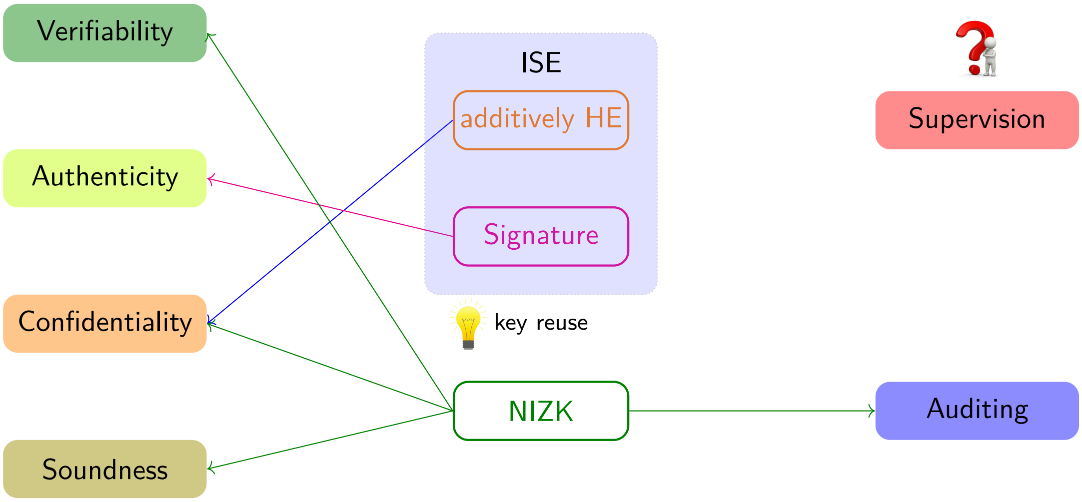
Choice of Building Blocks



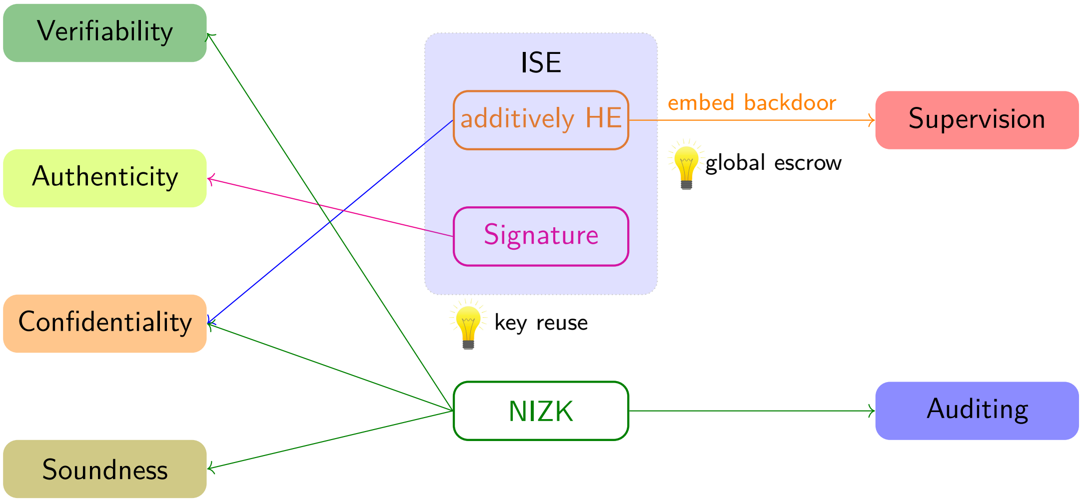
Choice of Building Blocks



Choice of Building Blocks



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 PPABC: 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 PPABC: 1/3

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

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

CreateAccount($\tilde{v}$): 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 PPABC: 2/3

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

1. $C_s \leftarrow \mathsf{ISE.Enc}(pk_s, v; r_1)$, $C_r \leftarrow \mathsf{ISE.Enc}(pk_r, v; r_2)$, $C_a \leftarrow \mathsf{ISE.Enc}(pk_a, v; r_3)$, memo $= (pk_s, pk_r, pk_a, C_s, C_r, C_a)$.
2. run NIZK.Prove with witness (sk_s, r_1, r_2, v) to generate a proof π_{legal} for memo $= (pk_s, pk_r, pk_a, C_s, C_r, C_a) \in L_{\mathsf{legal}} \mapsto L_{\mathsf{equal}} \wedge L_{\mathsf{right}} \wedge L_{\mathsf{solvent}}$

$$L_{\mathsf{equal}} = \{(pk_s, pk_r, pk_a, C_s, C_r, C_a) \mid \exists r_1, r_2, v \text{ s.t. } C_s = \mathsf{ISE.Enc}(pk_s, v; r_1) \wedge C_r = \mathsf{ISE.Enc}(pk_r, v; r_2) \wedge C_a = \mathsf{ISE.Enc}(pk_a, v; r_3)\}$$

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

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

3. $\sigma \leftarrow \mathsf{ISE.Sign}(sk_s, (\mathsf{memo}, \pi_{\mathsf{legal}}))$
4. output ctx $= (\mathsf{memo}, \pi_{\mathsf{legal}}, \sigma)$.

Algorithms of PPABC: 3/3

VerifyCTx(ctx): check if ctx is legal.

1. parse ctx $= (\text{memo}, \pi_{\text{legal}}, \sigma)$, memo $= (pk_s, pk_r, pk_a, C_s, C_r, C_a)$:
   1. check if ISE.Verify$(pk_s, (\text{memo}, \pi_{\text{legal}}), \sigma) = 1$;
   2. check if NIZK.Verify$(crs, \text{memo}, \pi_{\text{legal}}) = 1$.
2. ctx is recorded on the ledger if legality test passes or discarded otherwise.

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

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

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

OpenCTx(sp, ctx, sp): supervisor parses ctx $= ((pk_s, C_s, pk_r, C_r, pk_a, C_a), \text{aux})$, output ISE.Dec$(sp, C_a)$.

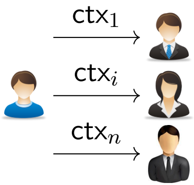
Regulation

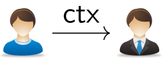
expressiveness of NIZK in use $\rightsquigarrow$ supported regulation policies

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

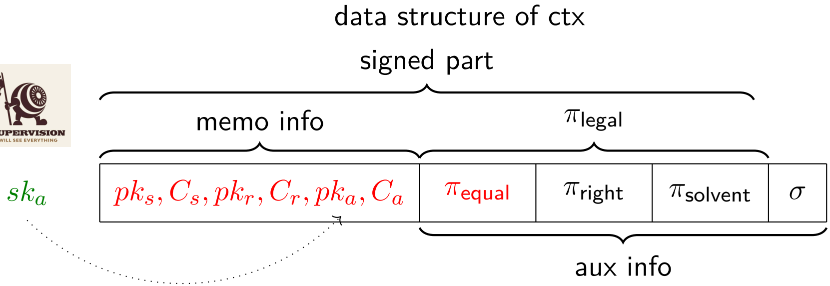
$f_{\text{rate}} : v_1 / v_2 = \rho$
pay tax

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





Supervision



Supervision

data structure of ctx

signed part: memo info + π_{legal}

aux info: $\pi_{\text{equal}}, \pi_{\text{right}}, \pi_{\text{solvent}}, \sigma$

SUPERVISION WILL SEE EVERYTHING

sk_a

memo info	π_{legal}			
$pk_s, C_s, pk_r, C_r, pk_a, C_a$	π_{equal}	π_{right}	π_{solvent}	σ

STOC 1990

Naor-Yung double enc paradigm → PKE CPA $\rightsquigarrow$ CCA

Supervision

data structure of ctx

signed part

memo info | π_{legal}

sk_a

$pk_s, C_s, pk_r, C_r, pk_a, C_a$	π_{equal}	π_{right}	π_{solvent}	σ

aux info

triple enc paradigm $\Longrightarrow$ auditor's view $=$ recipient's view

ensure supervision correctness

extend

Naor-Yung double enc paradigm $\longrightarrow$ PKE CPA $\rightsquigarrow$ CCA

STOC 1990

Generic Construction of PPABC

$$\mathsf{Setup}(1^\lambda) \rightarrow (pp, sp)$$

$$\mathsf{ISE.Setup}(1^\lambda) \rightarrow pp_{\mathsf{ise}},\ \mathsf{NIZK.Setup}(1^\lambda) \rightarrow pp_{\mathsf{nizk}}$$

$$\mathsf{ISE.Gen}(pp_{\mathsf{ise}}) \rightarrow (pk_a, sk_a)$$

embed backdoor for supervision

Generic Construction of PPABC

$\mathsf{Setup}(1^\lambda) \to (pp, sp)$

$\mathsf{ISE.Setup}(1^\lambda) \to pp_{\mathsf{ise}}$, $\mathsf{NIZK.Setup}(1^\lambda) \to pp_{\mathsf{nizk}}$
$\mathsf{ISE.Gen}(pp_{\mathsf{ise}}) \to (pk_a, sk_a)$

$\mathsf{CreateAccount}(v_s)$

$\mathsf{ISE.Gen}(pp_{\mathsf{ise}}) \to (pk_s, sk_s)$
$\mathsf{ISE.Enc}(pk_s, v_s) \to \tilde{C}_s$

$pk_s, sk_s, \tilde{C}_s$

$\mathsf{CreateAccount}(v_r)$

$\mathsf{ISE.Gen}(pp_{\mathsf{ise}}) \to (pk_r, sk_r)$
$\mathsf{ISE.Enc}(pk_r, v_r) \to \tilde{C}_r$

$pk_r, sk_r, \tilde{C}_r$

Generic Construction of PPABC

$$\mathsf{Setup}(1^\lambda) \rightarrow (pp, sp)$$

$$\mathsf{ISE.Setup}(1^\lambda) \rightarrow pp_{\mathsf{ise}},\ \mathsf{NIZK.Setup}(1^\lambda) \rightarrow pp_{\mathsf{nizk}}$$
$$\mathsf{ISE.Gen}(pp_{\mathsf{ise}}) \rightarrow (pk_a, sk_a)$$

$$\mathsf{CreateCTx}(pk_s, sk_s, pk_r, v) \rightarrow \mathsf{ctx}$$

$$\mathsf{ISE.Enc} \rightarrow \mathsf{memo} = (pk_s, C_s, pk_r, C_r, pk_a, C_a)$$
$$\mathsf{NIZK.Prove} \rightarrow \pi_{\mathsf{legal}} = \pi_{\mathsf{equal}} \circ \pi_{\mathsf{right}} \circ \pi_{\mathsf{solvent}}$$
$$\mathsf{ISE.Sign}(sk_s, (\mathsf{memo}, \pi_{\mathsf{legal}})) \rightarrow \sigma$$

$$\mathsf{CreateAccount}(v_s)$$

$$\mathsf{ISE.Gen}(pp_{\mathsf{ise}}) \rightarrow (pk_s, sk_s)$$
$$\mathsf{ISE.Enc}(pk_s, v_s) \rightarrow \tilde{C}_s$$

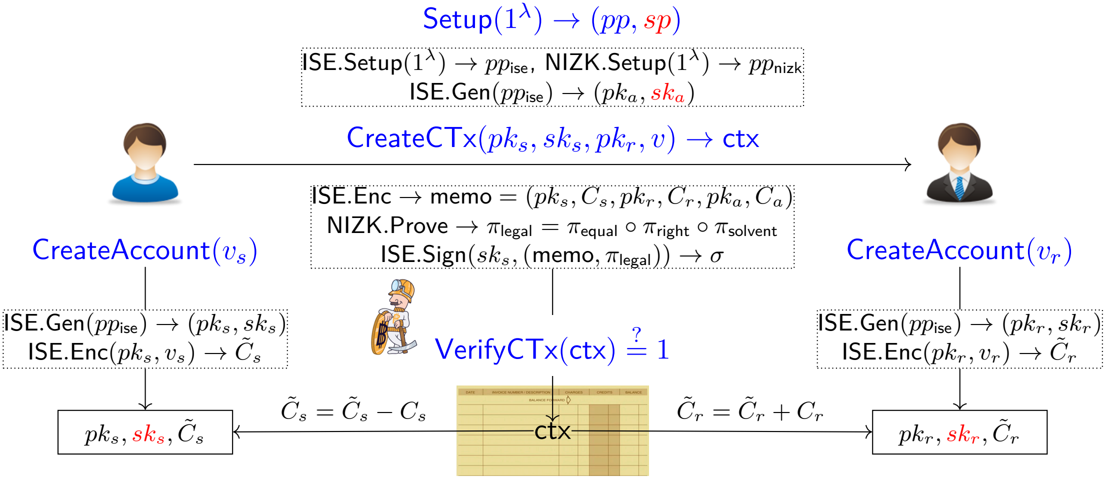
$$pk_s, sk_s, \tilde{C}_s$$

$$\mathsf{CreateAccount}(v_r)$$

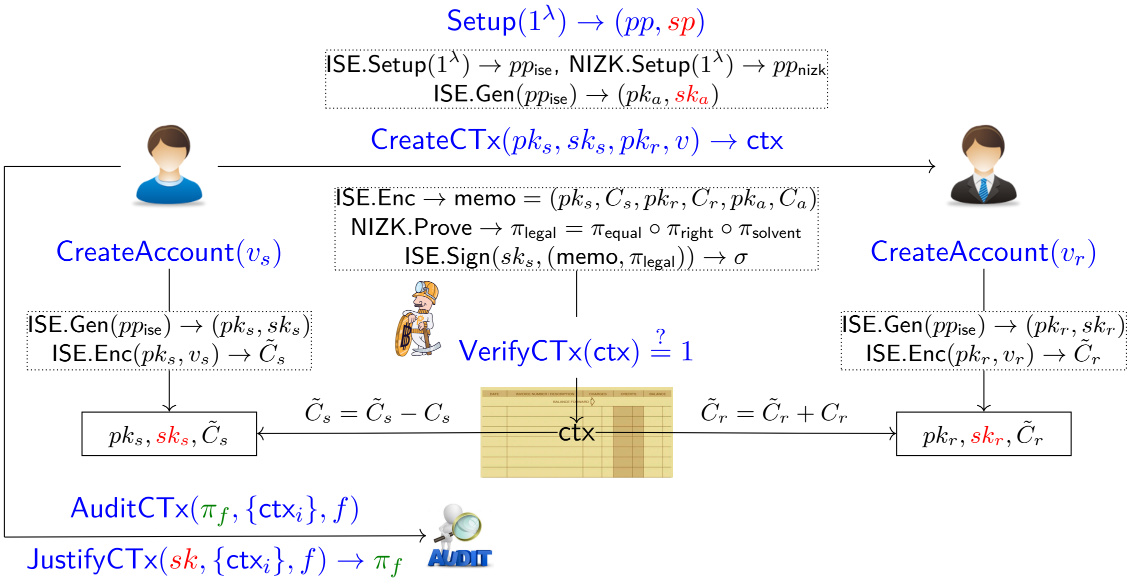
$$\mathsf{ISE.Gen}(pp_{\mathsf{ise}}) \rightarrow (pk_r, sk_r)$$
$$\mathsf{ISE.Enc}(pk_r, v_r) \rightarrow \tilde{C}_r$$

$$pk_r, sk_r, \tilde{C}_r$$

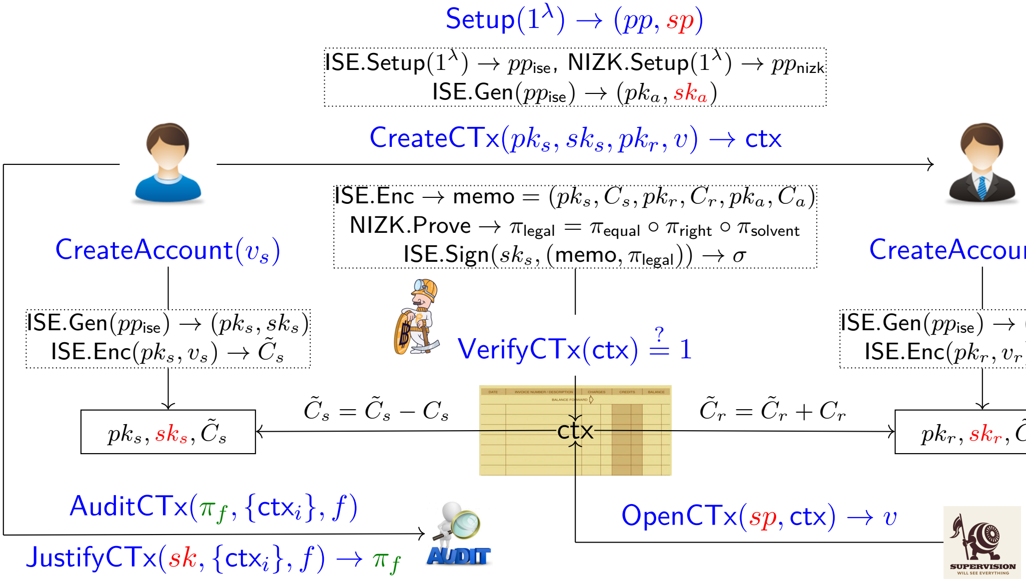
Generic Construction of PPABC



Generic Construction of PPABC



Generic Construction of PPABC



Security Proof

Theorem: Assuming the security of ISE and NIZK, our PPABC framework is secure.

- security of ISE's signature component $\Rightarrow$ authenticity
- security of ISE's PKE component + adaptive ZK of NIZK $\Rightarrow$ confidentiality
- adaptive soundness of NIZK $\Rightarrow$ soundness

Outline

Disciplines in Mind

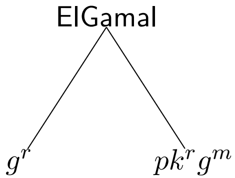
While PPABC framework is intuitive, secure and efficient instantiation requires clever choice and design of building blocks.

---

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

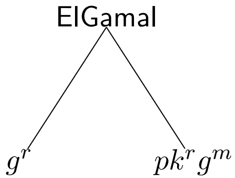
Encryption Component of ISE

the initial attempt



Encryption Component of ISE

the initial attempt

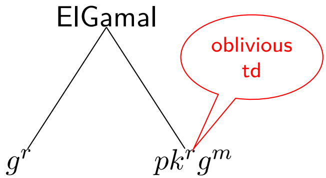


state-of-the-art

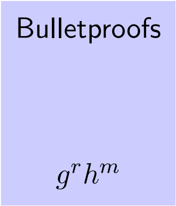
Bulletproofs

Encryption Component of ISE

the initial attempt

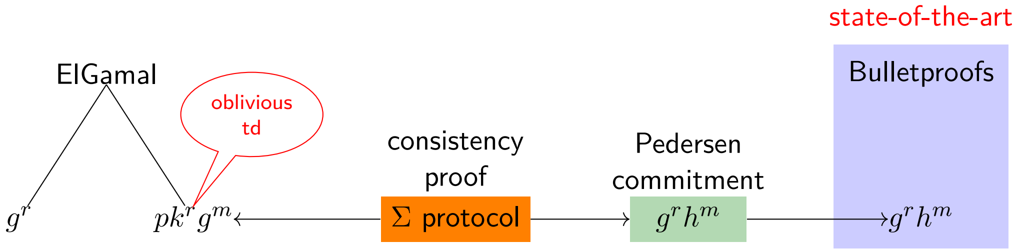


state-of-the-art



Encryption Component of ISE

the initial attempt

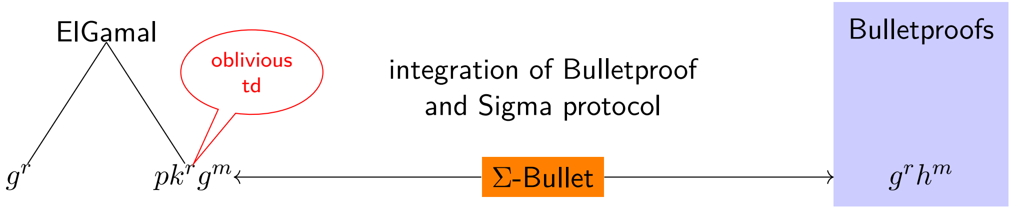


Quisquis's approach [FMMO19]
bring extra bridging cost

Encryption Component of ISE

the initial attempt

state-of-the-art



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

the initial attempt

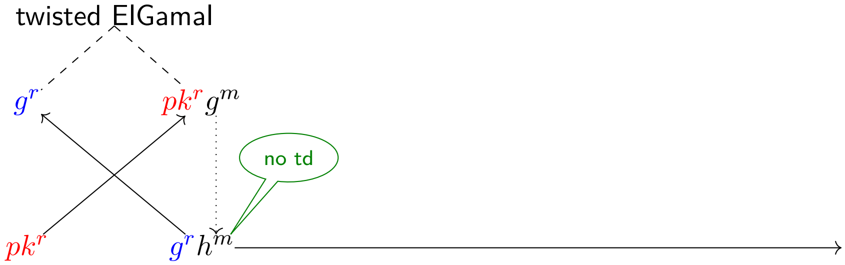
simple and efficient, but not friendly to the state-of-the-art range proofs

Encryption Component of ISE: Twisted ElGamal

twisted ElGamal

g^r $pk^r g^m$

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
  3. also friendly to other range proofs [CCS08, CKLR21] that accept Pedersen commitment as instance

Comparison to ElGamal

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

Related works [FMMO19, BAZB20] use brute-force algorithm to decrypt, we use Shanks's algorithm to accelerate 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	$2\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p\rvert$	4Exp+1Add
twisted	0	0

the saving could be tremendous when processing millions of data

Comparison to Paillier

Table: Twisted ElGamal vs. 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	5.4s+2.2s	0.009	0.094	0.239	0.099	0.003	0.003	0.08

with 64MB lookup table to accelerate decryption $4 \sim 300\times$ speedup in computation

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

$10\times$ speedup in communication

Details of Engineering Implementation

Standard Shanks algorithm: $\#\text{babystep} = \#\text{giantstep} = 2^{n/2}$.

- Trade space for time: set $\#\text{babystep} = 2^{n/2+r}$ and $\#\text{giantstep} = 2^{n/2-r}$ for better efficiency.

Lookup table is huge due to key is ECPoint

- Reduce the size of lookup table by using digest of ECPoint as key (at least 4 times smaller)

Push everything to the extreme

- Shanks's algorithm is highly parallelizable: using multithreading to speed
- store the reusable auxliary info to looktable to acceralate decryption

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.

   key reuse strategy ✓

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 $\sigma = (A, z)$.

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

joint security ✓

We can also use ECDSA/SM2 signature schemes.

NIZK for L_{equal}

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

$$\{(pk_i, X_i, Y_i)_{i\in[3]} \mid \exists r_1, r_2, r_3, v \text{ s.t. } X_i = pk_i^{r_i} \wedge Y_i = g^{r_i}h^v \text{ for } i = 1, 2, 3\}.$$

On statement $(pk_i, X_i, Y_i)_{i\in[3]}$, P and V interact as below:

1. P picks $a_1, a_2, a_3, b \xleftarrow{R} \mathbb{Z}_p$, sends $A_i = pk_i^{a_i}$, $B = g^{a_i}h^b$ to V.
2. V picks $e \xleftarrow{R} \mathbb{Z}_p$ and sends it to P as the challenge.
3. P computes $z_i = a_i + er_i$ for $i \in [3]$ and $t = b + ev$ using $w = (r_1, r_2, r_3, v)$, then sends (z_1, z_2, z_3, t) to V. V accepts iff the following four equations hold simultaneously:

$$pk_i^{z_i} = A_i X_i^e \quad (1)$$
$$g^{z_i}h^t = B_i Y_1^e \quad (2)$$

NIZK for L_{right}

Plug twisted ElGamal into PPABC framework, 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}

Σ_{enc} and Λ_{bullet} are acturally PoK + 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}

Plug twisted ElGamal into PPABC 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:

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

Prove it as L_{right}? No! $\boxed{r'}$ 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 Gadgets

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

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

Bonus: Two Useful Gadgets

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

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

---

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

prover is the sender of C

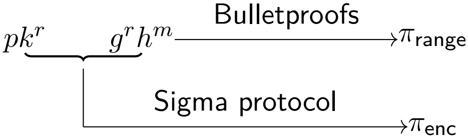
knows both r and m

Bonus: Two Useful Gadgets

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

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

prover is the sender of C
knows both r and m



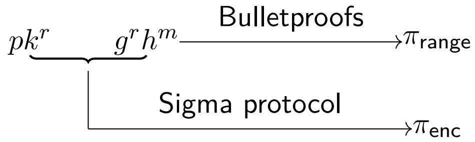
Bonus: Two Useful Gadgets

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

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

---

prover is the sender of C
knows both r and m



---

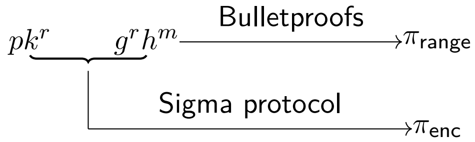
prover is the receiver of C
knows sk and thus m

Bonus: Two Useful Gadgets

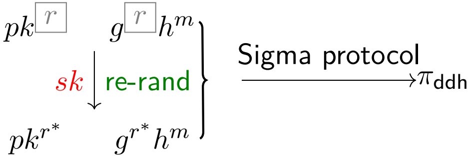
twisted ElGamal + Bulletproofs: prove an encrypted message lies in specific range

- 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



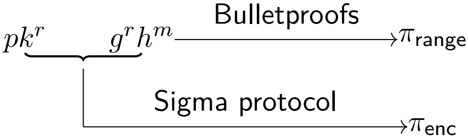
Bonus: Two Useful Gadgets

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

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

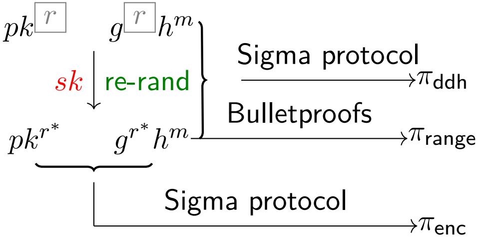
---

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)

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

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

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

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

NIZK for Auditing Policies: (2/2)

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

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

Thanks to nice algebra structure of twisted ElGamal, PGC supports efficient auditing for any policy that can be expressed as *linear constraints* over transfer amount and balance

Optimizations

$pk_s, C_s, pk_r, C_r, pk_a, C_a$	$\pi_{\mathsf{equal}} \circ (\pi^1_{\mathsf{enc}} \circ \pi^1_{\mathsf{bullet}}) \circ (C^* \circ \pi_{\mathsf{ddh}} \circ \pi^2_{\mathsf{enc}} \circ \pi^2_{\mathsf{bullet}})$	σ

Optimizations

randomness reuse
↓

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

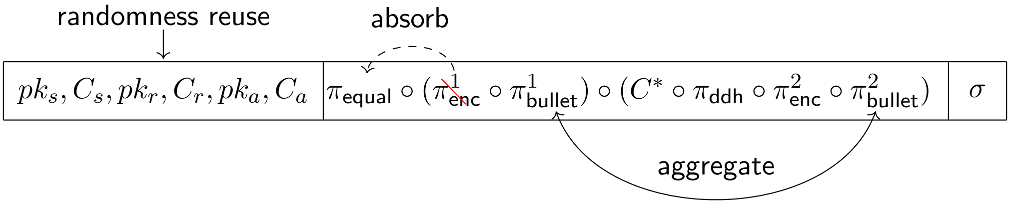
Randomness-Reusing

- original construction encrypts the same message v under pk_i $(i = \{s, r, a\}$ using independent random coins: $(pk_s, pk_s^{r_1}, g^{r_1}h^v, pk_r, pk_r^{r_2}, g^{r_2}h^v, pk_a, pk_a^{r_3}, g^{r_3}h^v)$
- twisted ElGamal is IND-CPA secure in 1-message/3-recipient setting

even when reusing randomness $\Rightarrow (pk_s, pk_s^r, pk_r, pk_r^r, pk_a, pk_a^r, \boxed{g^r h^v})$

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

Optimizations

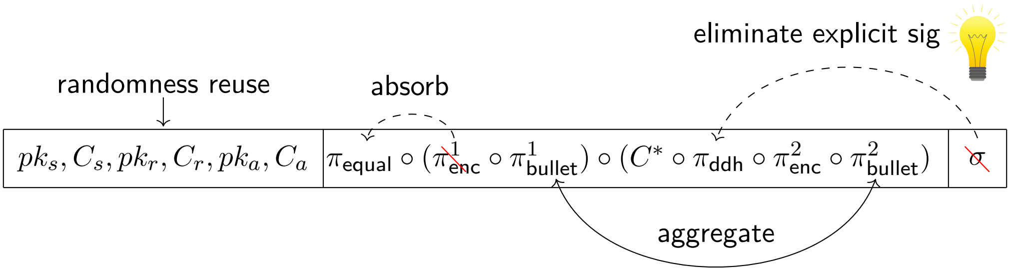


More Efficient Assembly of NIZK

- 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



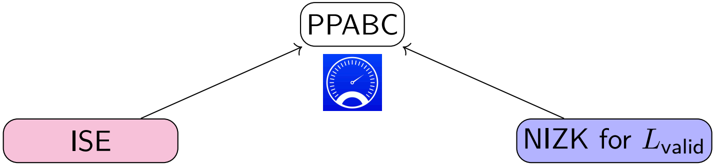
Eliminate Explicit Signature

- Σ_{ddh} (3-move public-coin ZKPoK of sk_s) is a sub-protocol of NIZK for L_{solvent}
- apply the Fiat-Shamir transform by appending the rest part to hash input $\rightsquigarrow \pi_{\text{ddh}}$ serves as both a proof of DDH tuple and a sEUF-CMA signature of ctx

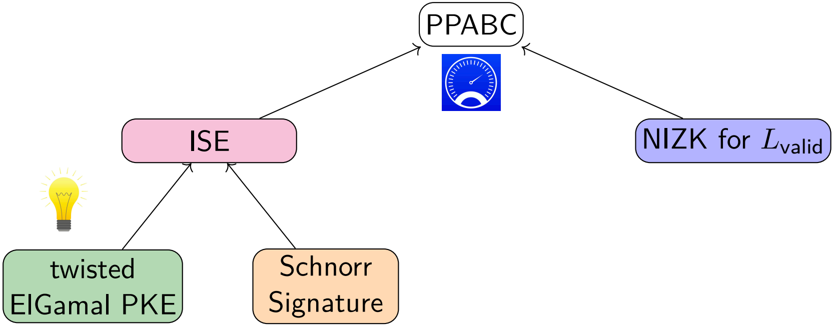
(still jointly secure with twisted ElGamal)

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

Recap of Efficient Instantiation



Recap of Efficient Instantiation



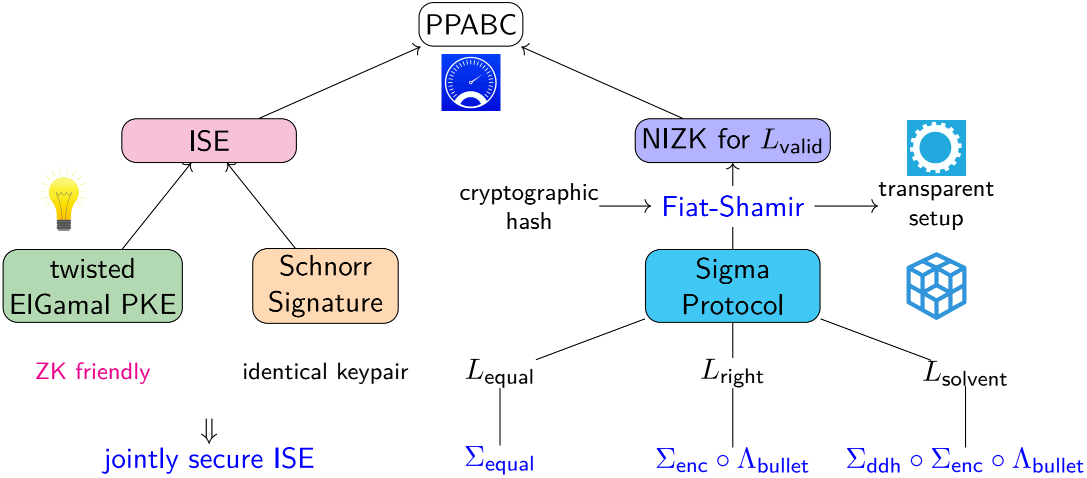
ZK friendly

identical keypair

$\Downarrow$

jointly secure ISE

Recap of Efficient Instantiation



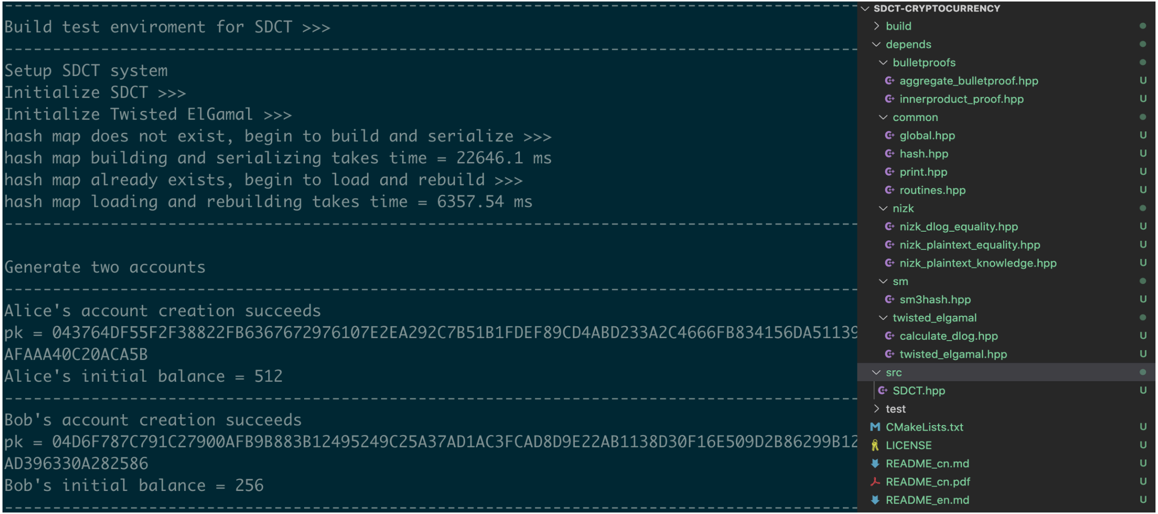
Outline

Deploy as a Standalone Cryptocurrency

Table: The computation and communication complexity of PGC

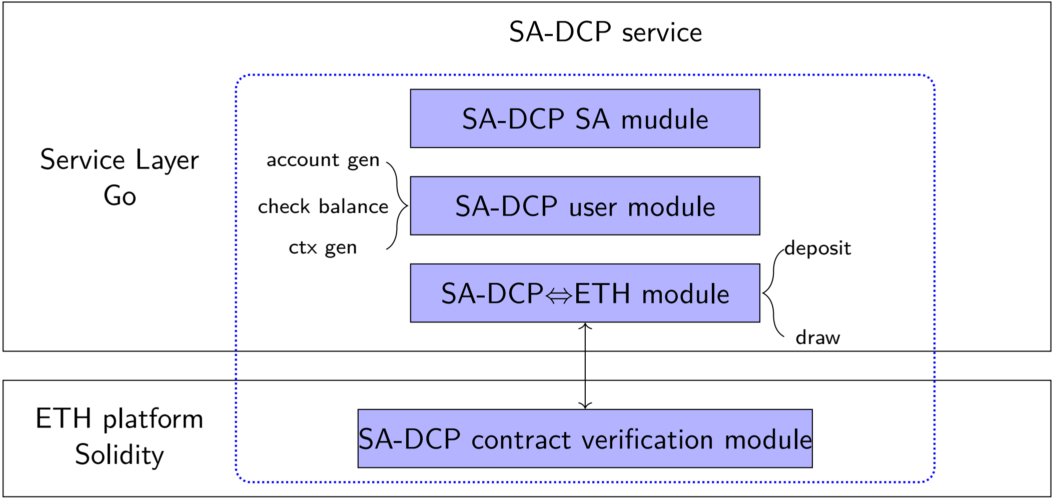
ADCP	ctx size		transaction cost (ms)	
	big-$\mathcal{O}$	bytes	generation	verify
transaction	$(2\log_2(\ell)+22)\lvert\mathbb{G}\rvert+11\lvert\mathbb{Z}_p\rvert$	1408	42	15
auditing	**proof size**		**auditing cost (ms)**	
	big-$\mathcal{O}$	bytes	generation	verify
limit policy	$(2\log_2(\ell)+4)\lvert\mathbb{G}\rvert+5\lvert\mathbb{Z}_p\rvert$	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
supervision	opening $\leq$ 1ms			

- Set $v_{\max} = 2^\ell - 1$, where $\ell = 32$
- Choose EC curve `secp256r1` (128 bit security), $|\mathbb{G}| = 33$ bytes, $|\mathbb{Z}_p| = 32$ bytes.
- MacBook Pro [Intel i7-4870HQ CPU (2.5GHz), 16GB of RAM]

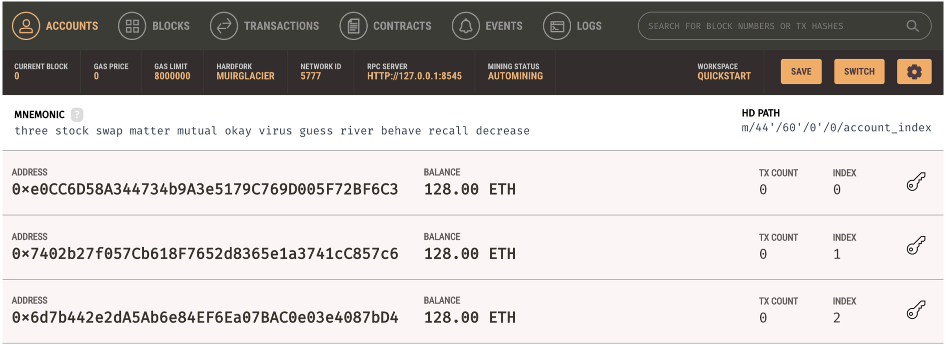

Build test enviroment for SDCT >>>
Setup SDCT system
Initialize SDCT >>>
Initialize Twisted ElGamal >>>
hash map does not exist, begin to build and serialize >>>
hash map building and serializing takes time = 22646.1 ms
hash map already exists, begin to load and rebuild >>>
hash map loading and rebuilding takes time = 6357.54 ms
Generate two accounts
Alice's account creation succeeds
pk = 043764DF55F2F38822FB6367672976107E2EA292C7B51B1FDEF89CD4ABD233A2C4666FB834156DA5113
AFAAA40C20ACA5B
Alice's initial balance = 512
Bob's account creation succeeds
pk = 04D6F787C791C27900AFB9B883B12495249C25A37AD1AC3FCAD8D9E22AB1138D30F16E509D2B86299B12
AD396330A282586
Bob's initial balance = 256
SDCT-CRYPTOCURRENCY
build
depends
bulletproofs
aggregate_bulletproof.hpp
innerproduct_proof.hpp
common
global.hpp
hash.hpp
print.hpp
routines.hpp
nizk
nizk_dlog_equality.hpp
nizk_plaintext_equality.hpp
nizk_plaintext_knowledge.hpp
sm
sm3hash.hpp
twisted_elgamal
calculate_dlog.hpp
twisted_elgamal.hpp
src
SDCT.hpp
test
CMakeLists.txt
LICENSE
README_cn.md
README_cn.pdf
README_en.md

Deploy as a Service

provide auditable confidential transaction service for ETH platform.



experimental result on ETH Ganache 2.4.0 $\rightsquigarrow$ SA-DCP service is practical




```
INFO[06-06|20:45:36] CTx transfer                              token=0x000000000000000000000000
amount=128 gas=2537804 tx=0xb8e319158cfd2996555a50f15e13069b811e85991d856a061abe00f4b5c8d4a3
--------------------------------------------------------------------------------
CTx transfer succeeds: Carol transfer 128 coins to Bob
Carol's current balance 0
Bob's current balance 256

INFO[06-06|20:45:43] CTx transfer                              token=0x000000000000000000000000
amount=128 gas=2499341 tx=0x58d522f75b0ab940ae0f47eb8c63bf85f89f9be5e4e035370cde0a45c3bc2416

CTx transfer succeeds: Bob transfer 128 coins to Alice
Bob's current balance 128
Alice's current balance 256
```

Outline

Comparison to Related Works

Table: Comparison to other acoount-based cryptocurrencies

Scheme	transparent setup	scalability	confidentiality	anonymity	regulation	supervision
zkLedger	✓+ DL	$O(n)$	?	✓	$O(m, \|f\|)$	✗
Zether	✓+ DL	$O(1)$	✓	✗	?	✗
PPABC	✓+ DL	$O(1)$	✓	✗	$O(\|f\|)$	✓

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) auditing efficiency is linear of both m and $|f|$ due to anonymity.
- Zether [BAZB20]: (i) Σ-Bullets require custom design, and its security is hard to check.
- In both zkLedger and Zether: (i) the confidentiality notion is not strong enough; (ii) signature and encryption are used in an adhoc manner, rather than in an integrated manner.

Summary

We propose a framework of PPABC from ISE and NIZK with formal security model and rigorous proof

- provide strong privacy and security guarantees for normal users
- provide handlers to conduct regulation and supervision for authority

We instantiate PPABC by carefully designing and combining cryptographic primitives $\rightsquigarrow$ PGC

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

Highlights

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

Global and Individual Supervision

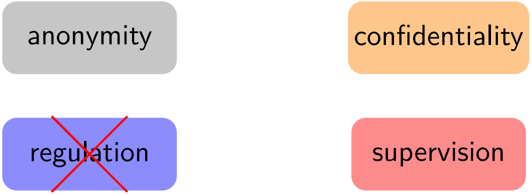
Supervision acturally comes with two flavors:

- Global supervision: A supervisor can inspect any transaction at his will.
  - This can be achieved by adpoting *global escrow* ISE. Naor-Yung paradigm used in this work happens to give a concrete instantiation.
- Individual supervision: A supervisor can inspect transactions associated to a specific user, which is more fine-grained than global supervision.
  - This can be achieved by adopting *hierarchy* ISE.

Yu Chen, Qiang Tang, Yuyu Wang
Hierarchical Integrated Signature and Encryption (or Key Separation vs. Key Reuse: Enjoy the Best of Both Worlds)
ASIACRYPT 2021

Ongoing Work

In our onoging work, we trade regulation for anonymity:



Prior work [Dia21] provides *limited* anonymity and does not support multi-receiver.

We construct fully-fledged PPABC that offers anonymity+confidentiality + supervision and supports multi-receiver based on newly introduced zero-knowledge proofs:

- k-out-of-n range proof
- inhomogeneous k-out-of-n proof

Min Zhang, **Yu Chen**, Xiyuan Fu, Zhiying Cui
k-out-of-n Proofs and Application to Privacy-Preserving Cryptocurrencies
ePrint 2025

Take Away

Crypto is not easy. Let alone using Crypto to build Crypto!

- Solid crypto foundation: provable security, all kinds of primitives and tools
- Profound computation science background
- Excellent programming skills

Take Away

Crypto is not easy. Let alone using Crypto to build Crypto!

- Solid crypto foundation: provable security, all kinds of primitives and tools
- Profound computation science background
- Excellent programming skills

Take Away

Crypto is not easy. Let alone using Crypto to build Crypto!

- Solid crypto foundation: provable security, all kinds of primitives and tools
- Profound computation science background
- Excellent programming skills

Exercise

How to prove two twisted ElGamal ciphertexts encrypt the same message?

Thanks for Your Attention!

Any Questions?

Reference I

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*, volume 12059, pages 423–443. Springer, 2020.

Jan Camenisch, Rafik Chaabouni, and Abhi Shelat.
Efficient protocols for set membership and range proofs.
In *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 234–252. Springer, 2008.

Geoffroy Couteau, Michael Klooß, Huang Lin, and Michael Reichle.
Efficient range proofs with transparent setup from bounded integer commitments.
In *Advances in Cryptology - EUROCRYPT 2021*, volume 12698 of *LNCS*, pages 247–277. Springer, 2021.

Reference II

- Benjamin E. Diamond.
  Many-out-of-many proofs and applications to anonymous zether.
  2021.

- 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.

- 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.