Epass: Efficient and Privacy-Preserving Asynchronous Payment on Blockchain

Weijie Wang, Jinwen Liang, *Member, IEEE,* Chuan Zhang, *Member, IEEE,* Ximeng Liu, *Senior Member, IEEE,* Liehuang Zhu, *Senior Member, IEEE*, and Song Guo, *Fellow, IEEE*

***Abstract*—Buy Now Pay Later (BNPL) is a rapidly proliferating e-commerce model, offering consumers to get the product immediately and defer payments. Meanwhile, emerging blockchain technologies endow BNPL platforms with digital currency transactions, allowing BNPL platforms to integrate with digital wallets. However, the transparency of transactions causes critical privacy concerns because malicious participants may derive consumers' financial statuses from on-chain asynchronous payments. Furthermore, the newly created transactions for deferred payments introduce additional time overheads, which weaken the scalability of BNPL services. To address these issues, we propose an efficient and privacy-preserving blockchain-based asynchronous payment scheme (Epass), which has promising scalability while protecting the privacy of on-chain consumer transactions. Specifically, Epass leverages locally verifiable signatures to guarantee the privacy of consumer transactions against malicious acts. Then, a privacy-preserving asynchronous payment scheme can be further constructed by leveraging time-release encryption to control trapdoors of redactable blockchain, reducing time overheads by modifying transactions for deferred payment. We give formal definitions and security models, generic structures, and formal proofs for Epass. Extensive comparisons and experimental analysis show that Epass achieves KB-level communication costs, and reduces time overhead by more than four times in comparisons with locally verifiable signatures and Go-Ethereum private test networks.**

***Index Terms*—E-commerce, locally verifiable signatures, time-released encryption, redactable blockchain.**

I. Introduction

Buy Now Pay Later (BNPL) allows consumers to pay in installments, usually several equal parts, and get the product immediately [1]. This service enables consumers to enjoy the products they need early even if they don't have enough money available, helping people to enhance their life experiences. Many shopping platforms, such as Amazon, Taobao, Jingdong, etc., support BNPL services using USD, RMB, etc. At the same time, the rise of blockchain has developed a new form of currency: digital currencies such as bitcoin and ether. According to statistics, as of early February 2023, users were paying a total of $4.4 million per day for Ethereum transactions [2]. Many businesses are also starting to support using digital currencies to shop for products. For example, Klarna allows customers to buy digital assets [3], and Affirm added cryptocurrency to allow customers to buy and sell digital currency on their application [4]. Therefore, it is not surprising to see that the BNPL platforms enable their customers to use digital currencies.

Existing works such as Pay Later Project (PLP) [5], Atpay [6], and Apenow [7], are performed by smart contracts to provide deferred payment services to consumers. Smart contracts allow for internal constraints requiring consumers to pay BNPL service providers at specific times [8]. However, the data transparency on-chain leads to serious privacy concerns [9], [10]. Transactions provided by consumers to smart contracts are visible to all nodes in the blockchain network, which also results in deferred payment transactions being as public as regular transactions. This is unacceptable in real-world e-commerce applications, as malicious third parties may analyze consumers' financial situation based on their deferred payment transactions [11]. In addition, it undermines the scalability of BNPL services due to the additional time overhead incurred by consumers' newly created deferred payment transactions. Each deferred payment transaction generated by a consumer appears as a new transaction in the blockchain network, and these newly generated transactions must again go through a series of processes, including acceptance, mining, propagation, and node network validation [12]. This procedure requires significant processing power and time and burdens the blockchain, especially for instalment payments.

Our goal is to build a services computing asynchronous payment system with the following features. First, it protects the privacy of users' transactions so that malicious third parties do not misuse their deferred payment transaction information. Second, it has a reasonable performance overhead and scalability as the number of users increases. An essential foundation for existing work to implement deferred payments is smart contracts, which monitor the behaviour of consumers and BNPL service providers through internal constraints. However, since smart contracts are public, all participants in the

Weijie Wang is with the School of Computer Science, Beijing Institute of Technology, Beijing, China. E-mail: weijiew@bit.edu.cn.

Jinwen Liang and Song Guo are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. E-mail: {jinwen.liang, song.guo}@polyu.edu.hk.

Chuan Zhang and Liehuang Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China. E-mail: {chuanz, liehuangz}@bit.edu.cn.

Ximeng Liu is with the College of Mathematics and Computer Science, Fuzhou University, and Fujian Provincial Key Laboratory of Information Security of Network Systems, Fuzhou, China. E-mail: snbnix@gmail.com.

Weijie Wang and Jinwen Liang contribute to the work equally and should be regarded as co-first authors.

Chuan Zhang is the corresponding author.

This research is financially supported by the "National Key R&D Program of China" (2021YFB2700500, 2021YFB2700503), the National Natural Science Foundation of China (Grant Nos. 6223000240, 62202051), the China Postdoctoral Science Foundation (Grant Nos. 2021M700435, and 2021TQ0042), the Shandong Provincial Key Research and Development Program (Grant No. 2021CXGC010106), and the Key-Area Research and Development Program of Guangdong Province under grant No.02021B0101400003.

Manuscript received April 19, 2021; revised August 16, 2021.

blockchain network can derive the input, execution process, and output of a smart contract [13]. Therefore, smart contracts will no longer meet our requirements because they are not suitable for protecting the privacy of users' transactions. In this work, we propose a deferred payment paradigm based on public key encryption with locally verifiable signatures to guarantee users' privacy. In addition, to preserve the scalability, we use the chameleon hash trap to modify the deferred payment transaction without creating additional new transactions, thus improving the efficiency of the blockchain. However, it is still challenging to enable BNPL service providers to make changes to transactions at specific times without using smart contracts. To address this issue, we combine timed-release encryption and a redactable blockchain to enable delayed payments. Specifically, we deploy servers that provide keys so that providers offering deferred payment services can only modify transactions at specific time intervals.

The main contributions of this paper.

- We propose a blockchain-based deferred payment scheme (Epass) that aims to address the privacy and efficiency issues in blockchain-authorized BNPL services. It reduces the on-chain burden while protecting user privacy.
- Epass has less time overhead compared to existing schemes. We deploy servers to provide time instant key so that the deferred payment service provider can only rewrite the on-chain transactions at a specific instant of time. Subsequent changes to the transaction through the trapdoor of chameleon hash only require node network verification, saving time and arithmetic power compared to generating a new transaction.
- Epass supports the protection of user privacy in the case of multiple transactions generated by users in the system. All transaction signatures of a user are aggregated into a single aggregated signature that is used for aggregation verification. We extend the single verification of locally verifiable signatures to subset verification. Later, without the service provider knowing all of the user's transactions, the subset of transactions requiring deferred payment can be decompressed from this aggregated signature.
- We have conducted extensive experiments on Epass. The results show that Epass has practical security features and acceptable communication cost (KB-level). Compared to the baseline, the time overhead was reduced by more than four times.

The remainder of this paper is organized as follows. Section II describes the symbols used and the cryptographic primitives involved. Section III describes the system architecture, threat model and security model. Section IV describes our proposed system in detail and gives a formal definition. Section V analyzes the correctness and security properties and performs a performance analysis in Section VI. Section VII describes the current issues in blockchain e-commerce solutions and gives a technical overview. Finally, we conclude our work in Section VIII.

II. Preliminary

In this section, we first define the notations to be used. Then we describe digital signatures, chameleon hashes and timed-release encryption.

A. Notation

The notations and corresponding descriptions in this paper are provided in Table I.

TABLE I
NOTATIONS

Notation	*Description*
$\mathbb{G}, \mathbb{G}_T$	bilinear groups
H, H_1, H_2	crytographic hash functions
mpk, msk	master key pair
pk_u, pk_m, pk_S	public key
pk_{local}	local verification key
sk_u, sk_m, sk_S	secret key
σ	signature
C	ciphertext
$\hat{\sigma}$	aggregated signature
tx	transaction
aux	auxiliaty information

B. Digital Signatures

Digital Signatures: A digital signature is a method of identifying digital information through public key cryptography to authenticate and confirm the identity and eligibility of the signer. Two complementary operations are typically defined by digital signatures, one for signing and the other for verifying. Sender $\mathcal{A}$ signs the data to be transmitted by its own private key to generate a digest, and then transmits the digest generated by the signature and the data to be transmitted together to receiver $\mathcal{B}$. Receiver $\mathcal{B}$ verifies the signature by $\mathcal{A}$'s public key after receiving the data. Four algorithms make up a digital signature DS [14], [15] with message space $\mathbb{M}$: $\{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}\}$.

- $\mathsf{DS.Setup}\left(1^{\lambda}\right) \longrightarrow (pp)$: Security parameter $\lambda \in \mathbb{N}$ is taken as an input, and a public parameter pp is output. Other algorithms take the public parameter pp as an implicit input.
- $\mathsf{DS.KeyGen}\,(pp) \longrightarrow (sk, pk)$: Following the entry of a public parameter pp, then provide a secret key sk and a public key pk.
- $\mathsf{DS.Sign}\,(sk, m) \longrightarrow (\sigma)$: A signature, denoted by the symbol σ, is produced upon the input of a secret key sk and a message $m \in M$.
- $\mathsf{DS.Verify}\,(pk, \sigma, m) \longrightarrow (b)$: A public key pk,a signature σ and a message $m \in M$ is input, and a decision bit $b \in \{0, 1\}$ is produced.

Existential Unforgeability Under a Chosen Message Attack (EU-CMA): Knowing the public key pk, a probability polynomial-time (probability polynomial-time, PPT) attacker is able to compute a valid signature for the new data M' with negligible probability after obtaining the valid digital signature it wishes to obtain. If a digital signature scheme satisfies the above security requirements, then a valid digital signature can convince the data receiver that the data it receives has not been tampered with and the sender of the data is the owner of the corresponding public key pk. Next, we give the security model of **EU-CMA**.

Definition 1. *(**EU-CMA** Security): On the subsequent experiment, the **EU-CMA** security definition of a digital signature* DS *is founded.*

$\mathbf{Exp}_{\mathcal{A},\mathsf{DS}}^{\mathrm{EUF-CMA}}\left(1^{\lambda}\right)$*:*

1) $pp \leftarrow$ ***DS****.Setup* $\left(1^{\lambda}\right)$;
2) $\mathcal{L}_{key} \leftarrow \emptyset$; *//KeyGen query list*
3) $\mathcal{L}_{sign} \leftarrow \emptyset$; *//Sign query list*
4) $\mathcal{L}_{corr} \leftarrow \emptyset$; *//Corrupt query list*
5) $(pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{KeyGen}(\cdot),\mathcal{O}_{Sign},\mathcal{O}_{Corrupt}(\cdot)}(pp)$;
6) *if* $pk^* \notin \mathcal{L}_{corr} \wedge (pk^*, m^*) \notin \mathcal{L}_{sign} \wedge$
7) ***DS****.Verify* $(pk^*, \sigma^*, m^*) = 1$,
8) *return* 1.
9) *else return* 0.

where

1) **Oracle** $\mathcal{O}_{KeyGen}(i)$*:*
2) $(sk, pk) \leftarrow \mathcal{DS}$. ***KeyGen*** (pp);
3) $\mathcal{L}_{key} \leftarrow \mathcal{L}_{key} \cup \{(i, sk, pk)\}$;
4) *return* pk.

1) **Oracle** $\mathcal{O}_{Sign}(pk, m)$*:*
2) $\sigma \leftarrow \mathcal{DS}.\mathrm{Sign}(sk, m)$;
3) $\mathcal{L}_{sign} \leftarrow \mathcal{L}_{sign} \cup \{(pk, m)\}$;
4) *return* σ.

1) **Oracle** $\mathcal{O}_{Corrupt}(pk)$*:*
2) $\mathcal{L}_{corr} \leftarrow \mathcal{L}_{corr} \cup \{(pk)\}$;
3) *return* sk.

When the following advantage is negligible for any probabilistic polynomial-time adversary $\mathcal{A}$, we claim that a digital signature scheme DS *is **EU-CMA** secure:*

$$\mathrm{Adv}_{\mathcal{A},DS}^{\textbf{EU-CMA}}\left(1^{\lambda}\right) = \left|\Pr\left[\mathrm{Exp}_{\mathcal{A},DS}^{\textbf{EU-CMA}}\left(1^{\lambda}\right) = 1\right]\right|.$$

C. Chameleon Hashes

Chameleon Hashes: Compared with the traditional hash function's difficulty in finding collisions, chameleon hash can artificially set a trapdoor, and mastering this trapdoor makes it easy to find hash collisions. To a certain extent, chameleon hash destroys the two collision resistance (weak collision resistance and strong collision resistance) characteristics of the hash function, and at the same time, it also destroys the tamper-evident property of the blockchain based on the hash function. But chameleon hash also expands the application scenarios of blockchain, and it remains infeasible for ordinary users who do not know the threshold to find collisions. In other words, the security of chameleon hash can also be guaranteed. For managers holding trapdoors, if they tamper with the blocks at will, it is also possible to verify whether the hashes of two blocks are equal. Five algorithms make up a chameleon hashes CH [16], [17] with message space $\mathbb{M}$: $\{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Verify}, \mathsf{Adapt}\}$.

- $\mathsf{CH.Setup}\left(1^{\lambda}\right) \longrightarrow (pp)$: Security parameter $\lambda \in \mathbb{N}$ is taken as an input, and a public parameter *pp* is output. Other algorithms take the public parameter *pp* as an implicit input.
- $\mathsf{CH.KeyGen}\,(pp) \longrightarrow (sk, pk)$: A public parameter *pp* is input, and a secret key sk and a public key *pk* are produced.
- $\mathsf{CH.Hash}\,(pk, m, r) \longrightarrow (h)$: A public key *pk*, a message $m \in \mathcal{M}$ and a randomness r is input, and a hash value h is produced.
- $\mathsf{CH.Verify}\,(pk, m, h, r) \longrightarrow (b)$: Following the entry of a public key *pk*, a message $m \in \mathcal{M}$, a hash value h and a randomness r, then provide a decision bit $b \in \{0, 1\}$.
- $\mathsf{CH.Adapt}\,(sk, m, h, r, m') \longrightarrow (r')$: Following the entry of a secret key *sk*, a message $m \in \mathcal{M}$, a hash value h, a randomness r and a message $m' \in \mathcal{M}$, then provide a randomness r'.

D. Timed-Release Encryption

Timed-Release Encryption: Timed-release encryption is a cryptographic primitive with a specific future decryption time specified by the sender. Its time-dependent features are important in many time-sensitive real-world applications (e.g., electronic bidding [18], installment payments [19], electronic confidential files [20]). The sender sends an encrypted message to the receiver, and no user, including the receiver, can decrypt the message until the specified time. Four algorithms make up a timed-release encryption TRE [21], [22] with message space $\mathcal{M}$ and time space $\mathcal{T} = [0, T-1]$: $\{\mathsf{Setup}, \mathsf{Ext}, \mathsf{Enc}, \mathsf{Dec}\}$.

- $\mathsf{TRE.Setup}\left(1^{\lambda}, T\right) \longrightarrow (pp)$: Security parameter $\lambda \in \mathbb{N}$ and a time instant T are taken as the input, and a public parameter *pp* is produced.
- $\mathsf{TRE.Ext}\,(mpk, msk, t) \longrightarrow (k_t)$: A master public key *mpk*, a master secret key *msk* and $t \in \mathcal{T}$ are input, and a time instant key (TIK) k_t is produced.
- $\mathsf{TRE.Enc}\,(mpk, m, [t_0, t_1]) \longrightarrow (C)$: A master public key *mpk*, a message $m \in \mathcal{M}$ and a Decryption Time Interval (DTI) $[t_0, t_1] \subseteq \mathcal{T}$ are input, and a ciphertext C is produced.
- $\mathsf{TRE.Dec}\,(mpk, C, k_t) \longrightarrow (m/\perp)$: Following the entry of a master public key *mpk*, a ciphertext C and a key k_t, then provide either a message m or a failure symbol $\perp$.

III. Problem Formulation

In this section, we first define the system model and then give detailed descriptions of the threat model and security model.

A. System Model

As shown in Figure 1, Epass is composed of a certificate authority (CA), users, providers, servers, and miners.

- Epass needs to be initialized, and CA is the blockchain administrator who must broadcast the system parameters to the other participants.
- Users are participants in the chain and they are allowed to choose between two types of payment forms. One is instant payment, and transactions generated in this way cannot be rewritten. The other is asynchronous payment, which allows the specified provider to rewrite the content of the transaction.

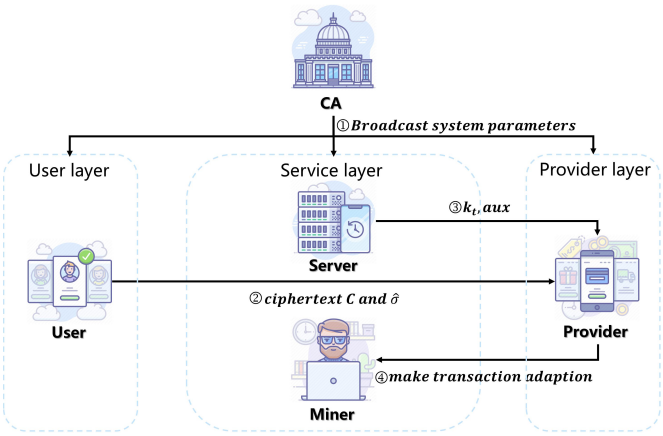

Fig. 1. System Model

- Providers are participants in the chain who provide asynchronous payment services to users and collect their fees when a specified time is reached.
- Servers are participants in the chain that broadcasts specific time nodes and provides additional auxiliary information to the provider.
- Miners are participants in the chain, independent and interconnected nodes that validate transactions and add them to the existing distributed public ledger.

B. Threat Model

In our proposed scheme, CA is considered to be fully trusted and miners are considered to be majority trusted, as in normal blockchain systems. The adversaries are classified into three categories based on their capabilities, i.e., intended-but-curious providers, honest-but-curious cloud and external adversary. The threat model is described in detail as follows.

- **Intended-but-Curious Providers**: The provider performs the role of a receiver and a provider in our system. As a semi-curious participant, the provider expects to decrypt messages outside of the expected time interval and tries to modify the wrong transaction amount.
- **Honest-but-Curious Cloud**: The cloud server holds the private key sk_s and is responsible for providing TIK k_t and auxiliary information aux at fixed intervals throughout the process. The cloud is semi-trustworthy and follows our protocol, but will attempt to launch attacks to compromise confidential messages, such as ciphertext-only attacks.
- **External Adversary**: An external attacker is neither the intended receiver nor aware of the sk_s, but he/she can eavesdrop during system communication and obtain the ciphertext by launching a ciphertext-only attack.

C. Security Model

We define a model for **IND-CPA** and **EU-CMA** security of Epass scheme.

Definition 2. *If the advantage of all adversaries in the game is negligible at most, then Epass is **IND-CPA** secure.*

Setup. *The challenger $\mathcal{C}$ runs Epass.$Setup(1^\lambda)$ to generate a master secret key msk and a master public key mpk. Then, the master secret key msk is given to the adversary $\mathcal{A}$.*

Phase 1. *At any time $t \in T$, $\mathcal{A}$ can adaptively issue a TIK extraction query to oracle. Oracle will respond to each query with k_t.*

Challenge. *$\mathcal{A}$ chooses two messages m_0 and $m_1 \in \mathcal{M}$ and a time interval $[t_0, t_1] \subseteq T$ with the constraint that for all queries t in Phase 1, $t \notin [t_0, t_1]$. $\mathcal{A}$ passes m_0, m_1, $[t_0, t_1]$ to $\mathcal{C}$. $\mathcal{C}$ chooses a random bit b and computes*

$$c' = \textit{Epass}.TrCreat\left(mpk, sk_u, (ID, tx_{ID}), sk_h, T\right).$$

c' is passed to $\mathcal{A}$.

Phase 2. *$\mathcal{A}$ continues to query the TIK extract oracle using the same restrictions as the Challenge phase.*

Guess. *The adversary outputs its guess b' for b.*

The output of this game is defined as 1 when $b' = b$ and 0 otherwise. If the output of the game is 1, we say that $\mathcal{A}$ succeeds. We denote the advantage of $\mathcal{A}$ winning the game by:

$$Adv_{\mathcal{A}}(\kappa) = \left|\Pr\left[b' = b\right] - \frac{1}{2}\right|.$$

*We solve the security problem of **IND-CCA** by extending the definition. First, consider a Decrypt oracle in addition. On input a pair (c, t), it gets the response kt after passing t to the TIK extraction oracle, where c represents the ciphertext and $t \in T$. Then, the Decrypt oracle returns a message m or a failure symbol $\perp$ to the adversary by running Epass.$ReleasedDec\left(\hat{\sigma}, pk_u, pk_S, m, aux, C, k_t\right)$. The Decrypt oracle can adaptively issue queries (c, t) in both Phase 1 and Phase 2, but will be restricted in the latter phase, i.e., the adversary cannot make decrypt queries (c, t), where $c = c'$, $t \in T$, c' represents the challenge ciphertext and T represents the time interval. Under this restriction, the adversary cannot win the game in a simple way.*

Definition 3. *(**EU-CMA** Security): If the advantage $\mathrm{Adv}^{\mathrm{EU-CMA}}_{\mathcal{A},Epass}(1^\lambda) = \mathrm{Adv}^{\textbf{EU-CMA}}_{\mathcal{A},textsfDS}(1^\lambda)$ is negligible for any probabilistic polynomial-time adversary $\mathcal{A}$, then Epass is **EU-CMA** secure. $\mathcal{O}$ is defined as the set of oracles, including: a provider key generation oracle $\mathcal{O}_{KeyGen_p}(\cdot)$, a provider corrupt oracle $\mathcal{O}_{Corrupt_p}(\cdot)$, a user key generation oracle $\mathcal{O}_{KeyGen_u}(\cdot)$, a user corrupt oracle $\mathcal{O}_{Corrupt_u}(\cdot)$, a server key generation oracle $\mathcal{O}_{KeyGen_S}(\cdot)$, a server corrupt oracle $\mathcal{O}_{Corrupt_S}(\cdot)$, a hash oracle $\mathcal{O}_{TrCreat}(\cdot,\cdot)$, an aggregate oracle $\mathcal{O}_{Aggregate}(\cdot,\cdot)$ and an adaption oracle $\mathcal{O}_{Adapt}(\cdot,\cdot,\cdot,\cdot,\cdot)$.*

D. Design Goals

Based on the requirements of the above models, our design goals are divided into two aspects: privacy and efficiency.

- *Privacy*: The privacy of user transactions should be guaranteed in asynchronous payments. The deferred payment transactions generated by the user should not be made public to prevent malicious third parties from analyzing the user's financial status accordingly.

- *Efficiency*: Reasonable performance overhead and scalability should be ensured. Epass saves time and computational power compared to existing works while satisfying basic transaction processing requirements.

IV. Privacy-Preserving Blockchain-based Asynchronous Payment Scheme

In this section, we give the formal definition and scheme description of Epass.

A. Formal Definition

Epass is made up of the ten algorithms listed below:

Epass.*Setup* $\left(1^{\lambda}\right) \rightarrow (pp, msk, mpk)$: CA manages the setup algorithm. It accepts a security parameter $\lambda \in \mathbb{N}$ as input. It produces a public parameter *pp*, a master secret key *msk*, and a master public key *mpk*, with *pp* and *mpk* serving as implicit inputs to all other algorithms.

Epass.*UserKeyGen* $(pp) \rightarrow (sk_u, pk_u)$: Each user executes the user setup algorithm. It accepts a public parameter *pp* as input and returns a secret key sk_u and a public key pk_u in the form of outputs.

Epass.*ProviderKeyGen* $(pp) \rightarrow \left(sk_p, pk_p\right)$: Each provider executes the provider setup algorithm. A secret key sk_p and a public key pk_p are produced from an input of a public parameter *pp*.

Epass.*ServerKeyGen* $(pp) \rightarrow (sk_S, pk_S)$: The server manages the server setup algorithm. A secret key sk_S and a public key pk_S are produced from an input of a public parameter *pp*.

Epass.*TrCreat* $(mpk, sk_u, (ID, tx_{ID}), sk_h, T) \rightarrow (h, r, \sigma, C)$: Each user executes the hash algorithm. The inputs are a message with a transaction identity *ID* and its content tx_{ID}, a master public key *mpk*, a pair of secret keys sk_u and sk_h, and a decryption time T. It generates a ciphertext C, a signature σ, a hash value h, and randomness r.

Epass.*Aggregate* $(pk_u, \{((ID_i, r_i), \sigma_i)\}_i) \rightarrow (\hat{\sigma}/\perp)$: All input signatures σ_i are first verified by the signature aggregation process, which outputs $\perp$ if any of these verifications are unsuccessful. If not, it produces the aggregated signature $\hat{\sigma}$.

Epass.*Ext* $\left(pk_S, sk_S, pk_u, \{tx_i\}_{i\in[\ell]}, j \in [\ell]\right) \rightarrow (k_t, aux)$: On the server, the extraction algorithm is executed. It accepts a collection of transactions $\{tx_i\}_{i\in[\ell]}$, public keys pk_S and pk_u, a secret key sk_S, and outputs a TIK k_t and the auxiliary information *aux*.

Epass.*ReleasedDec* $(\hat{\sigma}, pk_u, pk_S, m, aux, C, k_t) \rightarrow (sk_h/\perp)$: Each provider executes the timed-release decryption algorithm. It accepts the following as inputs: public keys pk_u and pk_S, an aggregate signature $\hat{\sigma}$, a message with a transaction identity *ID* and its content tx_{ID} and r, the auxiliary information *aux*, a ciphertext C, and the TIK k_t. And if any of these verification fail, outputs *bot*. If not, the aggregated signature sk_h is output.

Epass.*Adapt* $(mpk, sk_p, (ID, tx_{ID}), h, r, (ID, tx'_{ID})) \rightarrow (r', \sigma')$: Each provider executes the adapt algorithm. It accepts the following inputs: the master public key *mpk*, the secret key sk_p, the message's transaction identity *ID* and its content tx_{ID}, the hash value h, the randomness r, and the message's transaction identity *ID* and its content tx'_{ID}. It generates two values: randomness r' and a signature σ'.

Epass.*Verify* $(mpk, pk_u, (ID, tx_{ID}), h, r, \sigma_{ID}) \rightarrow (b)$: Miners operate the verification algorithm. A master public key *mpk*, a public key pk_u, a message including a transaction identity *ID* and its content tx_{ID}, a hash value h, a randomness r, and a signature σ_{ID} are all inputs required. It generates a judgment bit $b \in \{0, 1\}$ indicating the validity of the transaction (ID, tx_{ID}).

B. Proposed Scheme

The four phases of Epass are system initialization, transaction making, transaction rewriting, and transaction verification.

1) **System Initialization:** The initialization of Epass is displayed in Figure 2. This process can be more specifically divided into system setup, user key generation, provider key generation, and server key generation.

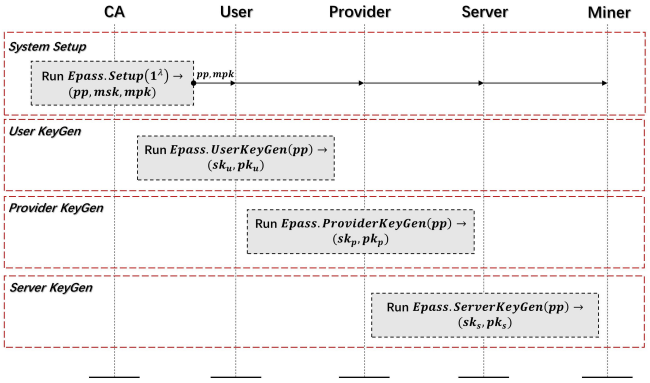

Fig. 2. System Initialization

Epass.*Setup* $\left(1^{\lambda}\right) \rightarrow (pp, msk, mpk)$: Given a security parameter $\lambda \in \mathbb{N}$ and the upper bound on number of aggregations B, set $pp_{\mathsf{DS}} = (p, \mathbb{G}, \mathbb{G}_T, g, \hat{e})$ as the bilinear group used in our construction, where $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, and $\mathbb{G}$ is a prime p order group. The public parameters of a chameleon hash is $pp_{\mathsf{CH}} \leftarrow \mathsf{CH}.Setup\left(1^{\lambda}\right)$. Selects a random number $\alpha \leftarrow \mathbb{Z}_p^*$, and samples the public parameters for message hashing as $hk \leftarrow HGen\left(1^{\lambda}\right)$. It sets the key pair as $pk_\alpha = (pp_{\mathsf{DS}}, \{g^{\alpha^i}\}_{i\in[B]})$, and $sk_\alpha = (pp_{\mathsf{DS}}, \alpha)$. Then, a collision-resistant hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$ is chosen, and the following cryptographic hash functions $H_1 : \{0,1\}^* \rightarrow G^*$, $H_2 : G_T^* \rightarrow \{0,1\}^n$ are constructed. The algorithm outputs a public parameter $pp = (pp_{\mathsf{DS}}, pp_{\mathsf{CH}}, hk)$, a master secret key $msk = sk_\alpha$, and a master public key $mpk = (pk_\alpha, H)$, where *pp* and *mpk* are made available to the public.

Epass.*UserKeyGen* $(pp) \rightarrow (sk_u, pk_u)$: The user key generation algorithm initializes a signature key-pair $sk_\beta = \left(pp_{\mathsf{DS}}, hk, \beta \leftarrow \mathbb{Z}_p^*\right)$, and $pk_\beta = (pp_{\mathsf{DS}}, hk, \{g^{\beta^i}\}_{i\in[B]})$, where B represents the upper bound for deferred payment transactions. It then initializes a chameleon hash key-pair $(sk_h, pk_h) \leftarrow \mathsf{CH.KeyGen}\left(pp_{\mathsf{CH}}\right)$, a timed-release

key pair $sk_{tre} = s \leftarrow \mathbb{Z}_p^*$, $pk_{tre} = (sg, s\alpha' g)$ and generate a local verification key $pk_{local} = (\Pi, \mathrm{hk}, g^{\alpha})$. The algorithm returns a secret key $sk_u = (sk_\beta, sk_{tre})$ and a public key $pk_u = \left(pk_\beta, pk_h, pk_{tre}, pk_{local}\right)$.

$\mathsf{Epass}.ProviderKeyGen\,(pp) \rightarrow \left(sk_p, pk_p\right)$: The provider key generation algorithm initializes a signature key-pair $sk_{p'} = (pp_{\mathsf{DS}}, hk, \mu \leftarrow \mathbb{Z}_p^*)$, and $pk_{p'} = (pp_{\mathsf{DS}}, hk, \{g^{\mu^i}\}_{i\in[B]})$. The algorithm outputs two keys: a secret one $sk_p = sk_{p'}$ and a public one $pk_p = pk_{p'}$.

$\mathsf{Epass}.ServerKeyGen\,(pp) \rightarrow (sk_S, pk_S)$: The server key generation algorithm initializes a key-pair as $sk_{\alpha'} = \left(pp_{\mathsf{DS}}, \alpha' \leftarrow \mathbb{Z}_p^*\right)$, and $pk_{\alpha'} = (pp_{\mathsf{DS}}, g \leftarrow \mathbb{G}, Z = \alpha' g)$, g and $\alpha' g$ are made public. The algorithm returns a secret key $sk_S = sk_{\alpha'}$ and a public key $pk_S = Z$.

2) **Transaction Making:** Figure 3 shows the Epass transaction making. Each user creates two kinds of transactions during this phase: redactable and immutable. The redactable transactions can be modified by the provider at a specific time. After that, the server generates TIK k_t and auxiliary information *aux*.

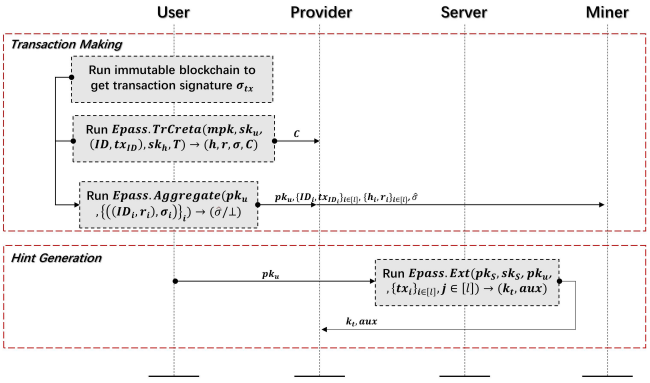

Fig. 3. Transaction Making

– **Ordinary and deferred transaction making:** The user can generate ordinary transactions as in a traditional immutable blockchain, or run $\mathsf{Epass}.TrCreat(mpk, sk_u, (ID, tx_{ID}), sk_h, T)$ to generate deferred transactions, as shown in Algorithm 1. In a normal transaction, the user generates a transaction *tx* and the corresponding signature σ_{tx} by means of the key sk_u and the traditional hash function. For deferred transactions, the user generates the transaction (ID, tx_{ID}) and the signature (h, r, σ_{ID}) by means of the key sk_u and the chameleon hash function. Finally, the aggregated signature $\hat{\sigma}$ is generated by running $\mathsf{Epass}.Aggregate(pk_u, \{((ID_i, r_i), \sigma_i)\}_i)$, as shown in Algorithm 2. The user propagates to the blockchain ecosystem the public key pk_p, the transaction set $\{tx_i\}_{i\in[\ell]}$, and the aggregated signature $\hat{\sigma}$.

– **Extraction:** At a time instant $T \in \{0,1\}^*$, the server publishes $k_t = \alpha' H_1(T)$, every user can verify its authenticity by checking $\hat{e}\left(\alpha' g, H_1(T)\right) = \hat{e}\left(g, \alpha' H_1(T)\right)$. Then, the server generates TIK k_t and auxiliary information aux by running $\mathsf{Epass}.Ext(pk_S, sk_S, pk_u, \{tx_i\}_{i\in[\ell]}, j \in [\ell])$, as shown in Algorithm 3. These information will be published at a specific time for the provider to verify the signature and perform decryption operations.

Algorithm 1: $\mathsf{Epass}.TrCreat(mpk, sk_u, (ID, tx_{ID}), sk_h, T)$

Input: Master public key *mpk*, secret keys sk_u and sk_h, a decryption time T, and a message containing the transaction identity *ID* and the transaction's content tx_{ID}.

Output: The randomness r, a hash value h, a randomness r, a signature σ and ciphertext C.

Choose a random number r;

Compute a hash value:

$$h \leftarrow \mathsf{CH}.Hash\left(pk_h, (ID, tx_{ID}), r\right);$$

Generate a signature: $\sigma_{ID} = g^{(1/sk_u + h)}$;

Check if $\hat{e}(sg, \alpha' g) \stackrel{?}{=} \hat{e}(g, s\alpha' g)$;

If so, random choose r_0, and compute $r_0 g$ and $r_0 s\alpha' g$;

Compute

$$K = \hat{e}\left(r_0 s\alpha' g, H_1(T)\right) = \hat{e}\left(g, H_1(T)\right)^{r_0 s\alpha'};$$

Compute the ciphertext:

$$C = < r_0 g, sk_h \oplus H_2(K) > .$$

return The hash value h, the random number r, the signature σ_{ID} and the ciphertext C.

Algorithm 2: $\mathsf{Epass}.Aggregate(pk_u, \{((ID_i, r_i), \sigma_i)\}_i)$

Input: Public key pk_u, signatures σ_i.

Output: The aggregated signature $\hat{\sigma}$.

if $e(\sigma, g^{\alpha} g^h) = e(g, g)$. **then**

  Compute: $\gamma_i = \frac{1}{\prod_{i\neq j}(x_i - x_j)}$,

  and compute the aggregated signature:

$$\hat{\sigma} = \prod_i \sigma_i^{\gamma_i}.$$

else

  return $\perp$.

3) **Transaction Rewriting:** Figure 4 shows the Epass transaction rewriting. In this phase, the provider decompresses the subset of transaction signatures that need to be paid asynchronously and performs the decryption operation based on the TIK k_t and auxiliary information aux provided by the server. The designated provider can rewrite the redactable transaction.

– **Timed-release decryption:** The provider holds the TIK k_t and auxiliary information *aux* provided by the server, and gets the secret key sk_h and the subset of transactions that need to be rewritten by running the $\mathsf{Epass}.ReleasedDec\left(\hat{\sigma}, pk_u, pk_S, m, aux, C, k_t\right)$, as shown in Algorithm 4. After that, the provider can use this secret key to rewrite the transaction.

– **Transaction rewriting:** The provider runs the

Algorithm 3: Epass.*Ext*$(pk_S, sk_S, pk_u, \{tx_i\}_{i\in[\ell]})$

Input: Public keys pk_S and pk_u, a secret key sk_S, a set of transactions $\{tx_i\}_{i\in[\ell]}$.

Output: The TIK k_t and the auxiliary information *aux*.

if $\hat{e}(\alpha' g, H_1(T)) = \hat{e}(g, \alpha' H_1(T))$. **then**

Generate the auxiliary informations:

$$aux_{j,1} = g^{\prod_{i\neq j}(\alpha+h_i)},$$

$$aux_{j,2} = g^{\alpha\prod_{i\neq j}(\alpha+h_i)};$$

Compute the following polynomial P to obtain the coefficients $\{\tilde{\delta}_i \in \mathbb{Z}_p\}_{i\in[\ell-1]}$:

$$P_{\{x_i\}_{i\in[\ell]\setminus\{j\}}}(y) = \prod_{i\in[\ell]\setminus\{j\}}(y+x_i) = \sum_{i=0}^{\ell-1}\tilde{\delta}_i y^i (\mathrm{mod} p);$$

Compute

$$aux_{j,1} = \prod_{i=0}^{\ell-1}(g^{\alpha^i})^{\tilde{\delta}_i},$$

$$aux_{j,2} = \prod_{i=0}^{\ell-1}(g^{\alpha^{i+1}})^{\tilde{\delta}_i};$$

Outputs the auxiliary informations

$$aux_j = (aux_{j,1}, aux_{j,2}).$$

else

return Null

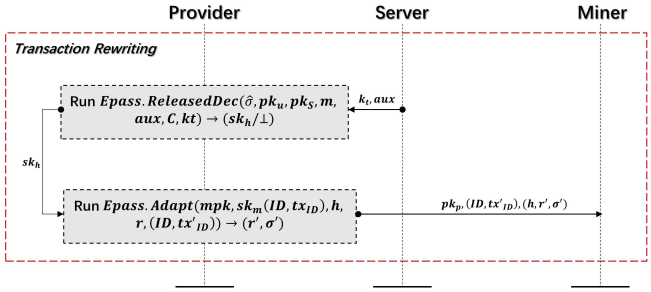

Fig. 4. Transaction Rewriting

Epass.*Adapt* $(mpk, sk_p, (ID, tx_{ID}), h, r, (ID, tx'_{ID}))$ algorithm to generate random numbers r' and signatures σ', as shown in Algorithm 5. Then, the provider broadcasts the public key pk_p, the transaction (ID, tx'_{ID}) and the signature (h, r', σ').

4) **Transaction Verification:** Figure 5 shows the Epass transaction verification. Users can create two different types of blockchain transactions, as was previously stated. Therefore, we consider two types of verification at this stage: immutable and redactable transaction verification.

Algorithm 4: Epass.*ReleasedDec*$(\hat{\sigma}, pk_u, pk_S, m, aux, C, k_t)$

Input: Public keys pk_u and pk_S, an aggregate signature $\hat{\sigma}$, a message containing the transaction's identity *ID* and content tx_{ID} and r, the auxiliary information *aux*, a ciphertext C and the TIK k_t.

Output: The aggregated signature sk_h or $\perp$.

Compute the message hash set as:

$$\{h_m\}_i \leftarrow \mathsf{CH}.Hash(pk_h, \{m\}_i);$$

if $e(\hat{\sigma}, aux_1^{\{h_m\}_i}\, aux_2) \stackrel{?}{=} e(g,g)$ *and* $e(g^\alpha, aux_1) \stackrel{?}{=} e(g, aux_2)$ **then**

return 1 to signal that the signature is valid;

Parse C as

$$< r_0 g, \rho = sk_h \oplus H_2(K) >;$$

Compute

$$K' = \hat{e}(r_0 g, \alpha' H_1(T))^s = \hat{e}(g, H_1(T))^{r_0 s\alpha'} = K;$$

Recover sk_h by computing $\rho \oplus H_2(K)$.

return sk_h

else

return $\perp$

Algorithm 5: Epass.*Adapt*$(mpk, sk_p, (ID, tx_{ID}), h, r, (ID, tx'_{ID}))$

Input: The master public key *mpk*, a secret key sk_p, a message including a transaction identity *ID* and its content tx_{ID}, a hash value h, a randomness r and a message including a transaction identity *ID* and its content tx'_{ID}.

Output: A randomness r' and a signature σ'.

Generate a randomness r':

$$r' \leftarrow \mathsf{CH}.Adapt\left(s\ k_h, (ID, tx_{ID}), h, r, (ID, tx'_{ID})\right);$$

Generate a signature: $\sigma' = g^{(1/sk'_p+h)}$.

return r' and signature σ'.

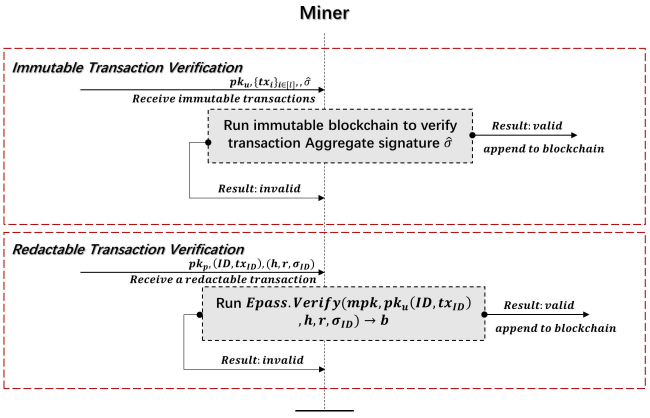

Fig. 5. Transaction Verification

- **Immutable transaction verification:** The miner receives a set of immutable transactions containing the user's public key pk_u, the transaction set $\{tx_i\}_{i\in[l]}$, and the aggregated signature $\hat{\sigma}$. The immutable blockchain verification procedure is used by the miner to validate the transaction. If the validation proceeds correctly, the miner accepts the related transaction and adds it to the block; if not, the transaction is rejected.
- **Redactable transaction verification:** After receiving a redactable transaction containing the user's public key pk_u, transaction ID, tx_{ID}, and signature h, r, σ_{ID}, the miner verifies the transaction by running $\mathsf{Epass}.Verify(mpk, pk_u, (ID, tx_{ID}), h, r, \sigma_{ID})$ algorithm to verify the transaction, as shown in Algorithm 6. The miner will update the local copy of the transaction if the validation is successful; otherwise, the miner will reject the transaction.

Algorithm 6: $\mathsf{Epass}.Verify(mpk, pk_u, pk_p, (ID, tx_{ID}), h, r', \sigma', \widehat{\sigma})$

Input: The transaction identity *ID* and its content tx_{ID}, the master public key *mpk*, a public key k_u, a hash value h, a randomness r, and a signature σ_{ID}.

Output: A a decision bit $b \in \{0, 1\}$.

if *the following conditions are true* **then**
  $\mathsf{CH.Verify}\left(pk_h, (ID, tx_{ID}), h, r'\right) = 1$
  $e(\widehat{\sigma}, \prod_{i=0}^{\ell}(g^{\alpha^i})^{\delta_i}) \stackrel{?}{=} e(g, g)$
  $e(\sigma', g^i g^{h'}) \stackrel{?}{=} e(g, g)$
  return $b = 1$
else
  return $b = 0$

V. Security Analysis

In this section, we demonstrate the security of the encryption algorithm and the unforgeability of the signature algorithm in Epass. Specifically, we analyze that Epass is **IND-CPA** and **EU-CMA** secure respectively.

Theorem 1. *Epass is **IND-CPA** secure, if the BDH assumption holds.*

Proof. We will prove that Epass is safe under the **IND-CPA** model by playing a game between the PPT adversary $\mathcal{A}$ and the simulator $\mathcal{S}$.

Setup: $\mathcal{S}$ replaces the master secret key with $(pp_{\mathsf{DS}}, \alpha_0)$, where α_0 is chosen randomly from $\mathbb{Z}_p^*$ and is unknown to $\mathcal{S}$. Here, both α and α_0 are random for $\mathcal{A}$, so $\mathcal{A}$ cannot distinguish between the real-world master secret key and the simulated one.

Phase 1: At any time $t \in T$, $\mathcal{A}$ can adaptively issue TIK extraction queries, and oracle will use k_t as the response to each query.

Challenge: $\mathcal{A}$ sends $m_0, m_1 \in \mathcal{M}$ and $[t_0, t_1] \subseteq T$ to $\mathcal{S}$. $\mathcal{S}$ chooses a random bit b and encrypts the message to get the challenge ciphertext c'. The simulation proceeds as follows:

1. $\mathcal{S}$ random choose r_0^*, and compute $r_0^* g$ and $r_0^* s\alpha' g$.
2. $\mathcal{S}$ compute
$$K = \hat{e}\left(r_0^* s\alpha' g, H_1(T)\right) = \hat{e}\left(g, H_1(T)\right)^{r_0^* s\alpha'}.$$
3. $\mathcal{S}$ compute the ciphertext:
$$c' = < r_0^* g, sk_h \oplus H_2(K) > .$$

Finally, $\mathcal{S}$ sends the challenged ciphertext c' to $\mathcal{A}$.

Phase 2: $\mathcal{A}$ continues to query the TIK extract oracle using the same restrictions as the Challenge phase.

Guess: $\mathcal{A}$ outputs its guess b' for b. When $b' = b$, $\mathcal{A}$ successfully wins the game.

Since b' is a random guess of $\mathcal{A}$, there is no advantage to $\mathcal{S}$ from $\mathcal{A}$'s guesses, so we can obtain the advantage that $\mathcal{A}$ undermines the security of our proposed scheme,

$$\mathrm{Adv}_{\mathcal{A}} \leq \frac{q-1}{2q}\epsilon_{\mathrm{BDH}}.$$

□

Theorem 2. *Epass is **EU-CMA** secure, which has the following advantage:* $\mathrm{Adv}^{\textbf{EU-CMA}}_{\mathcal{A},\textit{Epass}}(1^\lambda) = \mathrm{Adv}^{\textbf{EU-CMA}}_{\mathcal{A},\textit{DS}}(1^\lambda)$ *, if the underlying digital signature **DS** applied is **EU-CMA** secure.*

Proof. The standard security concept of our signature scheme is existential unforgeability under the choice message attack (**EU-CMA**) [23], which means that even if access to the signature oracle is gained, it is difficult to output a valid signature for a message m that has never been requested to the signature oracle. Epass is **EU-CMA** secure if no probabilistic polynomial-time adversary $\mathcal{A}$ can win with non-negligible probability. We construct simulator $\mathcal{B}$, which has a non-negligible probability to break the underlying signature scheme $\mathcal{C}$. The security game is defined as follows.

Setup:

$\mathcal{B}$ sends λ to $\mathcal{C}$ after receiving the security parameter $\lambda \in \mathbb{N}$, and $\mathcal{C}$ returns a public parameter pp_{DS}. $\mathcal{B}$ sends λ to $\mathcal{C}$ after receiving the security parameter $\lambda \in \mathbb{N}$, and $\mathcal{C}$ returns a public parameter pp_{DS}. $\mathcal{B}$ initializes the public parameter $pp_{\mathsf{CH}} \leftarrow \mathsf{CH}.Setup\left(1^\lambda\right)$, samples the public parameter of the message hash $hk \leftarrow HGen\left(1^\lambda\right)$, and $\mathcal{C}$ gives $\mathcal{B}$ the public key pk_α. Next, $\mathcal{B}$ sets the public parameters $pp = (pp_{\mathsf{DS}}, pp_{\mathsf{CH}}, hk)$ and the master public key $mpk = (pk_\alpha, H)$. Finally, $\mathcal{B}$ returns (pp, mpk) to $\mathcal{A}$ and gets lists:

$$\mathcal{L}_{\mathrm{KG_p}}, \mathcal{L}_{corr_{\mathrm{p}}}, \mathcal{L}_{\mathrm{KG_u}}, \mathcal{L}_{corr_{\mathrm{u}}}, \mathcal{L}_{\mathrm{KG_S}}, \mathcal{L}_{corr_{\mathrm{S}}}, \mathcal{L}_{\mathrm{h}}, \mathcal{L}_{agg}, \mathcal{L}_{apt} \leftarrow \emptyset.$$

Queries:

In this phase, $\mathcal{A}$ adaptively queries the following oracle.

$\mathcal{O}_{KeyGen_p}(sk_p, pk_p)$: $\mathcal{A}$ is allowed to query the provider key generation oracle. $\mathcal{C}$ provides $\{g^{\mu^i}\}_{i\in[B]}$ for $\mathcal{B}$. The algorithm returns a secret key $sk_p = (pp_{\mathsf{DS}}, hk, \cdot)$ and a public key $pk_p = (pp_{\mathsf{DS}}, hk, \{g^{\mu^i}\}_{i\in[B]})$. $\mathcal{B}$ updates the list

$$\mathcal{L}_{\mathrm{KG_p}} \leftarrow \mathcal{L}_{\mathrm{KG_p}} \cup \{(sk_p, pk_p)\}$$

and returns pk_p to $\mathcal{A}$.

$\mathcal{O}_{Corrupt_p}(pk_p)$: $\mathcal{A}$ can query the provider corrupt oracle using the message on pk_p. $\mathcal{B}$ sends the public key pk_p to $\mathcal{C}$ and gets

the corresponding private key sk_p. After that, $\mathcal{B}$ updates the list

$$\mathcal{L}_{corr_{\mathrm{p}}} \leftarrow \mathcal{L}_{corr_{\mathrm{p}}} \cup \{pk_p\}$$

and returns sk_p to $\mathcal{A}$.

$\mathcal{O}_{KeyGen_u}(pk_u)$: $\mathcal{A}$ is allowed to query the user key generation oracle. $\mathcal{B}$ receives pk_β, pk_h and pk_{tre} from $\mathcal{C}$ and immediately generates a local verification key $pk_{local} = (\Pi, hk, g^\alpha)$. The algorithm returns a private key $sk_u = (\cdot, \cdot)$ and a public key $pk_u = \left(pk_\beta, pk_h, pk_{tre}, pk_{local}\right)$. Finally, $\mathcal{B}$ updates the list

$$\mathcal{L}_{KG_{\mathrm{u}}} \leftarrow \mathcal{L}_{KG_{\mathrm{u}}} \cup \{pk_u\}$$

and returns pk_u to $\mathcal{A}$.

$\mathcal{O}_{Corrupt_u}(pk_u)$: $\mathcal{A}$ can query the user corrupt oracle using the message on pk_u. $\mathcal{B}$ sends pk_u to $\mathcal{C}$ and receives the corresponding sk_u. $\mathcal{B}$ updates list

$$\mathcal{L}_{corr_{\mathrm{u}}} \leftarrow \mathcal{L}_{corr_{\mathrm{u}}} \cup \{pk_u\}$$

and returns sk_u to $\mathcal{A}$.

$\mathcal{O}_{KeyGen_S}(pk_S)$: $\mathcal{A}$ is allowed to query the server key generation oracle. $\mathcal{B}$ receives pk_S from $\mathcal{C}$. The algorithm returns a private key $sk_S = (\cdot)$ and a public key pk_S. Finally, $\mathcal{B}$ updates the list

$$\mathcal{L}_{KG_{\mathrm{S}}} \leftarrow \mathcal{L}_{KG_{\mathrm{S}}} \cup \{pk_S\}$$

and returns pk_S to $\mathcal{A}$.

$\mathcal{O}_{Corrupt_S}(pk_S)$: $\mathcal{A}$ can query the server corrupt oracle using the message on pk_S. $\mathcal{B}$ sends pk_S to $\mathcal{C}$ and receives the corresponding sk_S. $\mathcal{B}$ updates list

$$\mathcal{L}_{corr_{\mathrm{S}}} \leftarrow \mathcal{L}_{corr_{\mathrm{S}}} \cup \{pk_S\}$$

and returns sk_S to $\mathcal{A}$.

$\mathcal{O}_{TrCreat}((pk_u, (ID, tx_{ID}), h, r, \sigma_{ID}))$: $\mathcal{A}$ can query the hash oracle using the message on pk_u, transaction tx_{ID} and its corresponding identity ID, hash value h, random number r and signature σ_{ID}. $\mathcal{B}$ selects a random number r and computes the hash $h \leftarrow \mathsf{CH}.Hash\left(pk_h, (ID, tx_{ID}), r\right)$. Next, if $pk_u \in \mathcal{L}_{corr_{\mathrm{u}}}$, a signature $\sigma_{ID} = g^{(1/sk_u+h)}$ is generated; otherwise, $\mathcal{B}$ sends the public key pk_u and the message (ID, r) to $\mathcal{C}$, and then gets σ_{ID} returned by $\mathcal{C}$. Finally, $\mathcal{B}$ updates the list

$$\mathcal{L}_{\mathrm{h}} \leftarrow \mathcal{L}_{\mathrm{h}} \cup \{(pk_u, (ID, tx_{ID}), h, r, \sigma_{ID})\}$$

and returns (h, r, σ_{ID}) to $\mathcal{A}$.

$\mathcal{O}_{Aggregate}(pk_u, \{((ID_i, r_i), \sigma_i)\}_i)$: $\mathcal{A}$ can query the aggregate oracle using the message on pk_u, and a transaction set $\{((ID_i, r_i), \sigma_i)\}_i$. $\mathcal{B}$ compute $\gamma_i = \frac{1}{\prod_{i \neq j}(x_i - x_j)}$, and $\hat{\sigma} = \prod_i \sigma_i^{\gamma_i}$ if $pk_u \in \mathcal{L}_{corr_{\mathrm{u}}}$, otherwise, $\mathcal{B}$ sends the public key pk_u and $\{((ID_i, r_i), \sigma_i)\}_i$ to $\mathcal{C}$, and then gets $\hat{\sigma}$ returned by $\mathcal{C}$. Finally, $\mathcal{B}$ updates the list

$$\mathcal{L}_{\mathrm{agg}} \leftarrow \mathcal{L}_{\mathrm{agg}} \cup \{pk_u, \{((ID_i, r_i), \sigma_i)\}_i\}$$

and returns $\hat{\sigma}$ to $\mathcal{A}$.

$\mathcal{O}_{Adapt}(pk_p, (ID, tx_{ID}), h, r, \sigma_{ID}, (ID, tx'_{ID}), r', \sigma'_{ID})$: $\mathcal{A}$ can query the hash oracle using the message on pk_u, transaction tx_{ID} and its corresponding identity ID, hash value h, random number r, signature σ_{ID}, transaction tx'_{ID} and its corresponding identity ID, new random number r', new signature σ'_{ID}. $\mathcal{B}$ generate a randomness $r' \leftarrow \mathsf{CH}.Adapt(sk_h, (ID, tx_{ID}), h, r, (ID, tx'_{ID}))$ and pick an index i that is never used before to generate a signature $\sigma' = g^{(1/sk'_p+h)}$ if sk_p has been corrupted, such that $pk_p \in \mathcal{L}_{corr_{\mathrm{p}}}$; otherwise, $\mathcal{B}$ receives a signature σ' form $\mathcal{C}$ after sending pk'_p and (ID, r') to $\mathcal{C}$. Finally, $\mathcal{B}$ updates the list

$$\mathcal{L}_{apt} \leftarrow \mathcal{L}_{apt} \cup \left\{\left(pk_p, (ID, tx_{ID}), h, r, \sigma_{ID}, (ID, tx'_{ID}), r', \sigma'_{ID}\right)\right\}$$

and returns r', σ'_{ID} to $\mathcal{A}$.

Output:

$\mathcal{A}$ can be passed a tuple

$$(pk^*, (ID^*, tx_{ID^*}), h^*, r^*, \sigma_{ID^*}, \widehat{\sigma}^*)$$

as input to $\mathsf{Epass}.Verify(pk^*, (ID, tx_{ID}), h^*, r^*, \sigma_{ID^*}, \widehat{\sigma}^*)$.

- $\mathcal{A}$ outputs a transaction that satisfies $pk^* = pk_u^* \wedge pk_u^* \notin \mathcal{L}_{corr_u} \wedge (pk_u^*, (ID^*, tx_{ID^*}), h^*, r^*, \sigma_{ID^*}) \notin \mathcal{L}_{\mathrm{h}}$ and is defined as a valid transaction, i.e., the transaction is uncorrupted and unqueried. Meanwhile, $\mathcal{A}$ can forge a signature σ^*_{ID} using sk_p to sign (ID^*, r^*). And $\mathcal{B}$ breaks the signature scheme by sending the key pk_u, the signature σ^*_{ID} and the message (ID^*, r^*) to $\mathcal{C}$.
- $\mathcal{A}$ outputs a message unrelated to the adaption oracle and a redacted transaction, i.e.

  $$pk^* = \left(pk_u^*, pk_p^*\right) \wedge \left(pk_p^*, \cdot, h^*, \cdot, \cdot, (ID^*, tx_{ID^*}), r^*, \sigma_{ID}\right),$$

  and thus

  $$pk^* \notin \mathcal{L}_{apt}.$$

  Meanwhile, $\mathcal{A}$ can forge a signature σ' using sk'_m to sign (ID^*, r^*). And $\mathcal{B}$ breaks the signature scheme by sending the key pk'_m, the signature σ' and the message (ID^*, r^*) to $\mathcal{C}$.

Therefore, through the above simulation process we can get the advantage

$$\mathrm{Adv}^{\textbf{EU-CMA}}_{\mathcal{A},\,\mathsf{Epass}}\left(1^\lambda\right) = \mathrm{Adv}^{\textbf{EU-CMA}}_{\mathcal{A},\mathcal{DS}}\left(1^\lambda\right).$$

□

VI. Performance

In this section, we evaluate and analyze the performance of Epass.

A. Settings

Setup. The operating environment we conducted our experiments was Windows 11, the system type was a 64-bit operating system, and the device was configured with an Intel(R) Core(TM) i5-10210U CPU @ 1.60 GHz 2.11 GHz and 8 GB of RAM on board. We used Java to implement the simulation, using the Java PairingBased Cryptography Library (JPBC) and Type A pairings to perform the Epass scheme. In the measurement, we use Java's function **Java.lang.System.currentTimeMillis()** to measure the running time from the start of the operation to the end of the operation. The experimental results are shown in Table II,

TABLE II
EXPERIMENTAL PERFORMANCES.

Algorithms	Time Cost in each algorithm(ms)[1][2](k = 8 / 16 / 24)			
	u = 8	u = 16	u = 24	u = 32
Syetem Setup	149 / 156 / 161	151 / 163 / 164	162 / 171 / 174	164 / 174 / 176
User Key Generation	440 / 463 / 494	438 / 460 / 495	439 / 461 / 496	440 / 464 / 495
Provider Key Generation	15 / 14 / 14	14 / 13 / 15	15 / 16 / 18	15 / 17 / 16
Server Key Generation	32 / 30 / 32	29 / 30 / 31	29 / 29 / 28	31 / 30 / 31

[1] k represents the number of deferred transactions for users.
[2] u represents the number of user.

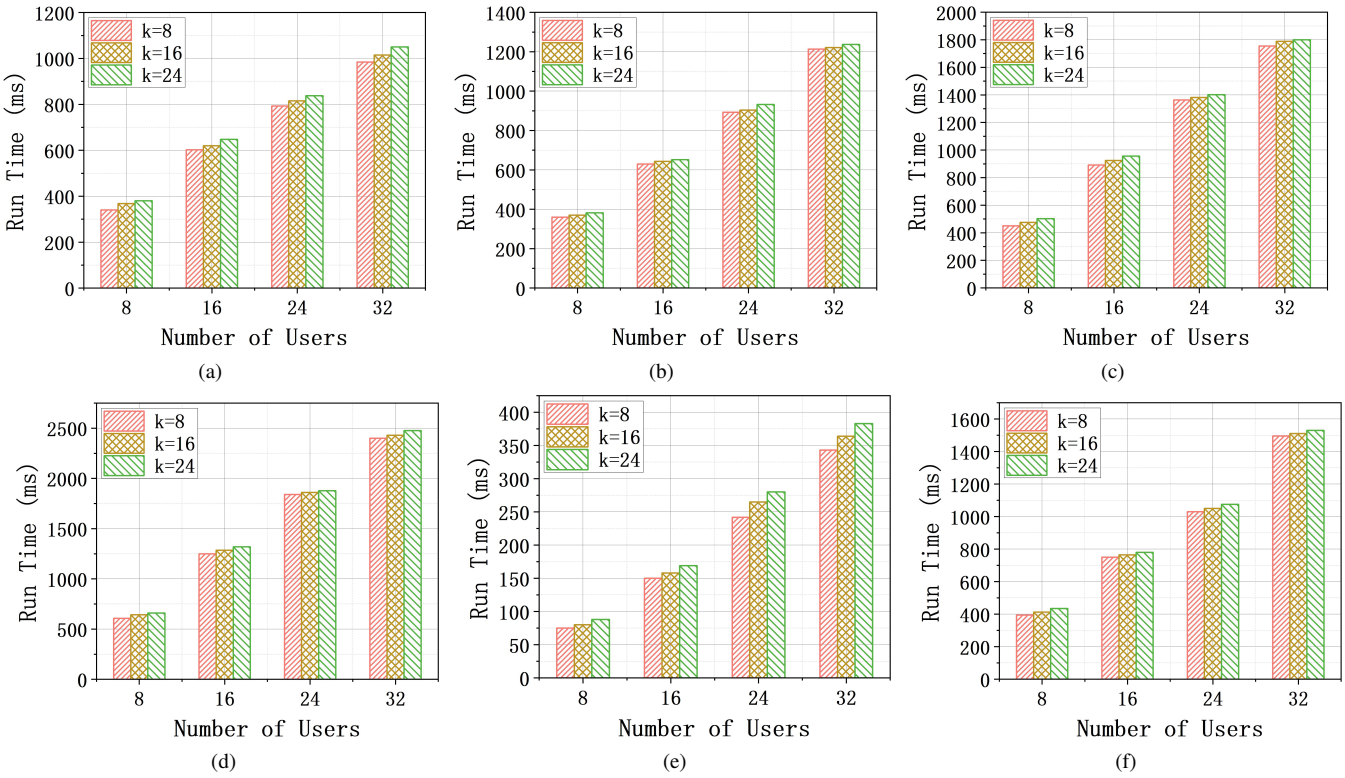

Fig. 6. Experimental performances. (a) time for transaction creation; (b) time for signature aggregation; (c) time for extraction; (d) time for timed-release decryption; (e) time for adaption; (f) time for verification. k represents the number of deferred transactions for users.

where k represents the number of deferred transactions for users.

Baselines for Comparison. We use the locally verifiable signature algorithm proposed in [24] (LVS) as a baseline. LVS can protect users' deferred payment information from being exploited by malicious third parties and reduce the time overhead of the system. However, LVS requires separate validation for each deferred payment transaction, which does not satisfy the efficiency of our design goals. Therefore, Epass extends the single verification in LVS to a subset of validations to reduce the time overhead.

Next, the most advanced implementation of BNPL schemes [5]–[7] handles deferred payment transactions via Ethernet smart contracts, so we use Geth to represent Ethernet smart contracts. Note that Geth implements deferred payments by generating new transactions, which somewhat undermines the scalability of the BNPL service. For this reason, Epass rewrites deferred payment transactions through the trapdoor of the chameleon hash [16], which does not generate additional new transactions during deferred payment and reduces the burden that Geth imposes on the blockchain.

Indicators. In the experiment, we evaluated the following indicators: i) *time costs:* the time cost of system initialization, transaction making, transaction rewriting, and transaction verification in Epass, and ii) *communication costs:* the size of the ciphertext generated by the user in the transaction-making phase.

B. Evaluation Results

Time costs. As shown in Table II, the time consumption in the system initialization phase (system setup, user key generation, provider key generation, and server key generation) is stable as the number of users and the number of deferred transactions increases. Among them, the primary time cost of the system initialization phase is concentrated on user key generation and system setup. With the number of users being 32 and the number of deferred transactions being 24, the time consumption of user key generation is less than 500ms, while the time consumption of system setup is less than 180ms.

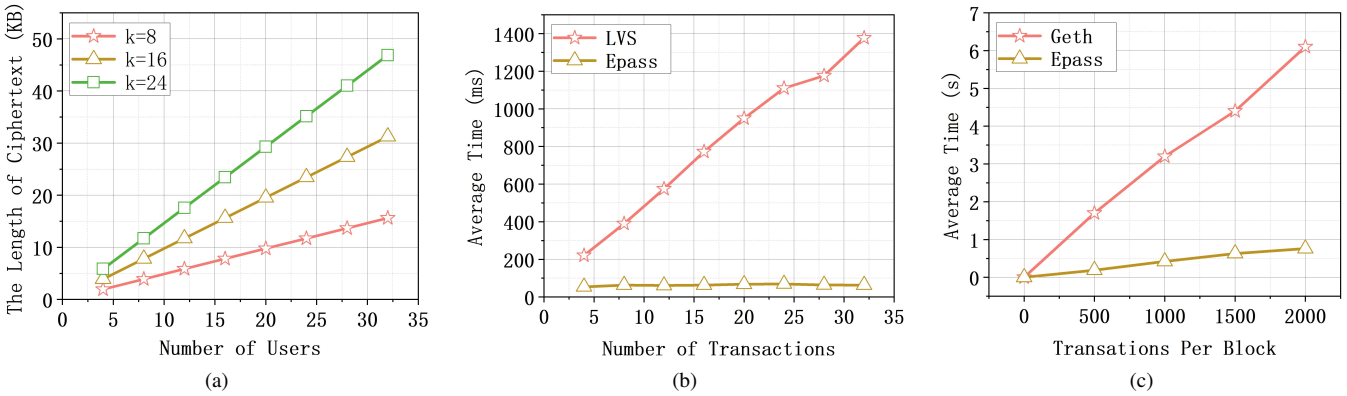

Fig. 7. Experimental performances. (a) time for process deferred transactions; (b) communication costs; (c) time for locally verification.

The time consumption of provider key generation and server key generation is negligible compared to the previous two phases. As the number of users and the number of deferred transactions increases, the time consumption of server key generation remains between 25 and 35 ms, while the time consumption of provider key generation is consistently below 20 ms. The results show that the time cost of Epass is acceptable during the system initialization phase.

The time consumption of transaction making (transaction creation, signature aggregation, and extraction), transaction rewriting (timed-release decryption and adaption), and transaction verification (verification) phases tend to increase linearly with the increase of input dimension. As shown in Figure 6, the input dimension appears as a set of users and a set of deferred payment transactions, and the size of the set depends on the number of users and deferred payment transactions, where k denotes the number of deferred payment transactions. The experimental results show that the primary time consumption comes from the transaction making phase. With the number of users at 32 and the number of deferred transactions at 24, the time consumption of the transaction making phase reaches 4000ms, of which transaction creation consumes about 1050ms, signature aggregation consumes about 1240ms, and extraction consumes about 1800ms. In the same input dimension, the time consumption of the transaction rewriting phase and transaction verification phase is 2860ms and 1500ms, respectively. In contrast, the time consumption of timed-release decryption, adoption, and verification is 2500ms, 380ms, and 1500ms, respectively. Moreover, the number of deferred transactions has less impact on the time consumption compared to the number of users. This is due to the fact that the operations involved in delayed transactions are usually exponential in nature and do not impose a significant burden on the computational overhead. In summary, the time cost of Epass is acceptable.

Communication costs. As shown in Figure 7(a), we use the size of the ciphertext generated by users in the transaction making phase to evaluate the communication cost for the different numbers of users and the different numbers of deferred transactions. Figure 7(a) shows that the communication costs increase linearly with the number of users and the number of deferred transactions. When the number of users reaches 32, and the number of deferred transactions reaches 24, the communication cost is still less than 50KB. The experimental results show that the number of users and the number of deferred transactions impact the communication cost. The communication cost of Epass is acceptable when appropriate parameters are chosen.

Comparison with baseline. Next, we extend LVS even further with the locally verifiable signature that supports subset verification. As shown in Figure 7(c), the time consumption of LVS grows linearly with the number of signatures to be verified. As the number of signatures to be verified increases from 4 to 32, the time consumption grows from 200ms to 1400ms. In comparison, when the number of signatures to be verified is 4 and 32, the time consumption of Epass to verify these signatures is less than 50ms and 70ms, respectively. The results show that the time cost of Epass is acceptable when dealing with local verification of signatures.

Finally, we created a private test network using Go-Ethereum and compared the time required to process deferred transactions between the baseline and Epass. We performed block creation experiments in which the block difficulty target was set to $0x0400$. This allowed us to accurately measure the operation time without increasing the work permit overhead while avoiding causing extreme blockchain forks. As shown in Figure 7(b), the time overheads of Epass and Geth both grow linearly. When transactions per block reach 2000, the time overhead of Epass is less than 1s, while Geth reaches 6s, which is six times higher than our scheme. The experimental results show that as the number of transactions in each block increases, Epass leads to better efficiency and is more suitable for real-world applications.

VII. Related Work

This section briefly reviews the background and related work on BNPL.

A. Background

Over the past few years, BNPL has gradually increased in popularity among financial institutions, merchants, and consumers as online shopping has proliferated in pandemic proportions [25], [26]. BNPL companies have created one of

the fastest-growing segments of consumer finance, according to GlobalData [27]. BNPL allows consumers to purchase a product immediately and pay for it over a period of time, usually in fixed installments [28]. It is a reverse-assist system where the purchaser can immediately get the product or service and then pay for it. Like how digital assets and blockchain technology have infiltrated most businesses, BNPL platforms enable their customers to use cryptocurrencies. Affirm [4], Zip [29], Klarna [3], and XRPayNet [30] are experimenting with their blockchains to create an application that offers BNPL functionality. Affirm's app brings all its products together in one place. At the same time, it's google chrome extension allows customers to use Affirm at various retailers, even if the service is not integrated into their checkout options [4]. Zip offers the same type of service as Affirm, allowing customers to buy now and then pay weekly or monthly, which is great flexibility for customers. In addition, Zip offers a digital wallet that can carry up to $1,000 and is interest-free [31]. Klarna works with more than 5,000 banks in 18 European countries by partnering with Swedish cryptocurrency broker Safello [32] to provide it with open banking infrastructure. Safello's 180,000 customers will now use Klarna's available banking payment system to buy cryptocurrencies without leaving Safello's platform [33]. XRPayNet allows businesses to continue using their existing processing systems, making the crypto-to-fiat payment process seamless [34]. As more and more people discover cryptocurrency, businesses are beginning to accept it as a payment method. It was only a matter of time before platforms combined it with BNPL options.

B. Blockchain-based Buy Now Pay Pater

The explosive growth of the e-commerce industry has provided consumers worldwide with multiple ways to purchase their favorite products while saving time and money. With the integration of options such as cryptocurrency and BNPL, customers and merchants alike are reaping additional benefits. However, only some e-commerce platforms offer these benefits to customers. But so far, there has been little discussion about blockchain-based BNPL. Until recently, only a few popular companies were offering this service through smart contracts, including PLP [5], Atpay [6], and Apenow [7]. There are two types of participants in these solutions, the user (consumer) and the merchant. Users on the blockchain-based BNPL platform can make purchases and earn rewards. Verified users can shop online and in-store from any platform-approved merchant. At checkout, users can select the BNPL feature to pay. When they complete their repayment, they receive rewards in the form of tokens that can also be used for future purchases or to earn higher purchase limits. Merchants on the blockchain-based BNPL platform can grow and expand their business and incentivize their customers with marketing campaigns. Using the blockchain-based BNPL service, users and merchants can benefit from it.

PLP [5] is a DEFI protocol and is the first BNPL platform built to integrate blockchain technology with its own cryptocurrency. While providing significant cost savings to all participants in the ecosystem by leveraging smart contract technology and blockchain, PLP will also allow users to pay for their purchases with any recognized cryptocurrency they hold in their PLP wallet.

Atpay [6] is merging blockchain and cryptocurrency technology with the BNPL concept, enabling consumers to shop online and offline and use specific cryptocurrencies when paying from the platform's native wallets. Customers get access to a wide range of payment options, significant cost savings, and rewards when they shop. The platform is also supported by its native cryptocurrency @Pay tokens, allowing holders to participate in the management of the agreement.

Apenow [7] is an NFT installment purchase agreement built on Teller that supports the ability of buyers to finance their next purchase. Users can make a down payment on an NFT purchase, and the remaining amount can be paid in installments. The platform holds the NFT in escrow until the full payment is completed, and the user can withdraw the NFT to their wallet only after the payment is completed. If the user does not complete the installments by the agreed deadline, the NFT will be liquidated and reimbursed to the loan provider.

However, these solutions do not consider the privacy issues arising from on-chain data transparency and the additional time overhead caused by deferred payments. Specifically, the transactions generated by users are visible to all nodes in the blockchain so that a malicious third party may analyze users' wealth status based on their deferred payment transaction information. Besides, deferred payment transactions generated by users incur additional time overhead, which undermines the scalability of the blockchain-based BNPL service. Therefore, in this work, we construct an efficient and privacy-preserving blockchain deferred payment solution to address the problems that exist in the current work.

VIII. Conclusion

This work investigates the privacy and efficiency issues associated with the BNPL model in the blockchain. To address these issues, we propose Epass, an efficient and privacy-preserving blockchain-based asynchronous payment scheme. By extending single verification of locally verifiable signatures to a subset of verification, Epass has lower time consumption while protecting consumer privacy. Asynchronous payments are implemented by leveraging timed-release encryption and redactable blockchain, saving time and arithmetic power compared to existing schemes. Extensive experiments and security analysis show that Epass has practical security features and acceptable communication cost (KB-level). The time overhead was reduced by more than four times compared to the baseline.

References

[1] "Buy now, pay later. what is the case with crypto?" 2022, https://blog.mercuryo.io/post/bnpl-crypto.

[2] "Crypto fees," 2023, https://cryptofees.info/.

[3] (2021) Klarna goes crypto: Bnpl fintech dips toes into crypto waters with safello partnership. https://www.altfi.com/.

[4] (2022) Affirm debuts app that will speed up planned crypto offering. https://www.digitalcommerce360.com/.

[5] "A defi buy now pay later ecommerce solution for consumers merchants," 2022, https://paylaterproject.com/.

[6] "Defi ecommerce payment solution for shoppers, merchants web3 businesses," 2022, https://atpay.io/.

[7] "Buy now, pay later for nfts has arrived," 2022, https://apenowpaylater.com/.

[8] D. Sinha and S. R. Chowdhury, "Blockchain-based smart contract for international business–a framework," *Journal of Global Operations and Strategic Sourcing*, 2021.

[9] P. Shah, D. Forester, M. Berberich, C. Raspé, and H. Mueller, "Blockchain technology: Data privacy issues and potential mitigation strategies," *Practical Law*, 2019.

[10] C. Zhang, L. Zhu, C. Xu, K. Sharif, K. Ding, X. Liu, X. Du, and M. Guizani, "Tppr: A trust-based and privacy-preserving platoon recommendation scheme in vanet," *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 806–818, 2022.

[11] S. Li, C. Xu, Y. Zhang, Y. Du, and K. Chen, "Blockchain-based transparent integrity auditing and encrypted deduplication for cloud storage," *IEEE Transactions on Services Computing*, vol. 16, no. 1, pp. 134–146, 2023.

[12] S. Gupta and M. Sadoghi, "Blockchain transaction processing," *arXiv preprint arXiv:2107.11592*, 2021.

[13] S. Li, Y. Zhang, C. Xu, N. Cheng, Z. Liu, Y. Du, and X. Shen, "Healthfort: A cloud-based ehealth system with conditional forward transparency and secure provenance via blockchain," *IEEE Transactions on Mobile Computing*, pp. 1–18, 2022.

[14] R. Kaur and A. Kaur, "Digital signature," in *2012 International Conference on Computing Sciences*, 2012, pp. 295–301.

[15] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology—CRYPTO'87: Proceedings 7*, 1988, pp. 369–378.

[16] H. Krawczyk and T. Rabin, "Chameleon hashing and signatures," *Cryptology ePrint Archive*, 1998.

[17] G. Ateniese and B. de Medeiros, "Identity-based chameleon hash and applications," in *Financial Cryptography: 8th International Conference, FC 2004, Key West, FL, USA, February 9-12, 2004. Revised Papers 8*, 2004, pp. 164–180.

[18] A. W. Dent and Q. Tang, "Revisiting the security model for timed-release encryption with pre-open capability," in *ISC*, vol. 4779, 2007, pp. 158–174.

[19] W. Mao, "Timed-release cryptography," *Cryptology ePrint Archive*, 2001.

[20] M. C. Mont, K. Harrison, and M. Sadler, "The hp time vault service: exploiting ibe for timed release of confidential information," in *Proceedings of the 12th international conference on World Wide Web*, 2003, pp. 160–169.

[21] G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan, "Conditional oblivious transfer and timed-release encryption," in *EuroCrypt*, vol. 99, 1999, pp. 74–89.

[22] J. Cathalo, B. Libert, and J.-J. Quisquater, "Efficient and non-interactive timed-release encryption," in *Information and Communications Security: 7th International Conference, ICICS 2005, Beijing, China, December 10-13, 2005. Proceedings 7*, 2005, pp. 291–303.

[23] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on computing*, vol. 17, no. 2, pp. 281–308, 1988.

[24] R. Goyal and V. Vaikuntanathan, "Locally verifiable signature and key aggregation," *Cryptology ePrint Archive*, 2022.

[25] A. Sengupta, "Bnpl as a new financial instrument its impact on consumer's buying behaviour," *Available at SSRN 4175857*, 2022.

[26] T. Muparadzi and L. Rodze, "Business continuity management in a time of crisis: emerging trends for commercial banks in zimbabwe during and post the covid-19 global crisis," *Open Journal of Business and Management*, vol. 9, no. 3, pp. 1169–1197, 2021.

[27] L. Feng, J.-T. Teng, and F. Zhou, "Pricing and lot-sizing decisions on buy-now-and-pay-later installments through a product life cycle," *European Journal of Operational Research*, vol. 306, no. 2, pp. 754–763, 2023.

[28] B. Guttman-Kenney, C. Firth, and J. Gathergood, "Buy now, pay later (bnpl)... on your credit card," *arXiv preprint arXiv:2201.01758*, 2022.

[29] (2021) Pay later where you want. https://www.reuters.com/.

[30] (2023) The world's most diverse payment system. https://xrpaynet.com/.

[31] C. Fisher, C. Holland, and T. West, "Developments in the buy now, pay later market," *1. 1 Cash Demand During COVID-19 2. 12 Property Settlement in RITS 3. 21 From the Archives: The London Letters 4. The Anatomy of a Banking Crisis: Household Depositors in the Australian 33 Depressions*, p. 59, 2021.

[32] (2023) Bye sek. hello crypto! https://safello.com/.

[33] (2021) Klarna teams up with safello - bringing open banking to cryptocurrency market. https://news.cision.com/safello/.

[34] (2022) Xrpaynet: Redefining crypto payments with diverse payment systems! https://www.platinumcryptoacademy.com/.

Weijie Wang received his B.S. degree from Xidian University in 2020. He is currently a master student in the School of Computer Science at Beijing Institute of Technology. His research interests include federal learning, security and privacy in blockchain.

Jinwen Liang is a Postdoctoral fellow in the Department of Computing of the Hong Kong Polytechnic University. He received his Ph.D. degree and B.S. degree from Hunan University, China, in 2021 and 2015, respectively. From 2018 to 2020, he was a visiting Ph.D. student at BBCR Lab, University of Waterloo, Canada. His research interests include applied cryptography, AI security, blockchain, and database security. He served as the Technical Program Committee Chair of the 1st international workshop on Future Mobile Computing and Networking for Internet of Things (IEEE FMobile 2022), Publicity Chair of the 6th International Workshop on Cyberspace Security (IWCSS 2022), TPC Member of IEEE VTC' 19 Fall. He is a member of the IEEE.

Chuan Zhang received his Ph.D. degree in computer science from Beijing Institute of Technology, Beijing, China, in 2021. From Sept. 2019 to Sept. 2020, he worked as a visiting Ph.D. student with the BBCR Group, Department of Electrical and Computer Engineering, University of Waterloo, Canada. He is currently an assistant professor at School of Cyberspace Science and Technology, Beijing Institute of Technology. His research interests include secure data services in cloud computing, applied cryptography, machine learning, and blockchain.

Ximeng Li (Senior Member, IEEE) received the B.S. degree in electronic engineering from Xidian University, Xi'an, China, in 2010 and the PhD degree in cryptography from Xidian University, China, in 2015. Currently, he is a full professor with the College of Mathematics and Computer Science, Fuzhou University, China. Also, he is a research fellow with the School of Information System, Singapore Management University, Singapore. His research interests include cloud security, applied cryptography and big data security.

Liehuang Zhu (Senior Member, IEEE) received his Ph.D. degree in computer science from Beijing Institute of Technology, Beijing, China, in 2004. He is currently a professor at the School of Cyberspace Science and Technology, Beijing Institute of Technology. His research interests include security protocol analysis and design, group key exchange protocols, wireless sensor networks, and cloud computing.

Song Guo (Fellow, IEEE) is a Full Professor in the Department of Computing at The Hong Kong Polytechnic University. He also holds a Changjiang Chair Professorship awarded by the Ministry of Education of China. His research interests are mainly in the areas of big data, edge AI, mobile computing, and distributed systems. With many impactful papers published in top venues in these areas, he has been recognized as a Highly Cited Researcher (Web of Science) and received over 12 Best Paper Awards from IEEE/ACM conferences, journals and technical committees. Prof. Guo is the Editor-in-Chief of IEEE Open Journal of the Computer Society. He has served on IEEE Communications Society Board of Governors, IEEE Computer Society Fellow Evaluation Committee, and editorial board of a number of prestigious international journals like IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Cloud Computing, IEEE Internet of Things Journal, etc. He has also served as chair of organizing and technical committees of many international conferences. Prof. Guo is an IEEE Fellow and an ACM Distinguished Member.