Tipping Pennies? Privately. Practical Anonymous Micropayments.

Bogdan Carbunar, Yao Chen, Radu Sion

***Abstract*—We design and analyze the first *practical* anonymous payment mechanisms for network services. We start by reporting on our experience with the implementation of a routing micropayment solution for Tor. We then propose micropayment protocols of increasingly complex requirements for networked services, such as p2p or cloud-hosted services.**

The solutions are efficient, with bandwidth and latency overheads of under 4% and 0.9 ms respectively in the ORPay implementation, provide full anonymity (for both payers and payees), and support thousands of transactions per second.

I. Introduction

With increasingly complex webs of networked, interacting entities, efficient payment mechanisms become paramount. And while traditional electronic payment and e-cash systems serve well for sizable cash amounts, they feature impractical overheads for small, penny-level payments, due to expensive infrastructure and protocol level constructs.

Yet, small online cash (or non-cash – e.g., quality of service – tokens) transactions are becoming increasingly popular. Users can download MP3 music from websites (e.g. iTunes store [1]) for tens of pennies. Providing network services such as routing [2] and P2P file sharing [3] feature sub-penny service costs per routed unit or shared file. In such settings, simple and efficient *micro*payment mechanisms are required with lower overheads than existing payment infrastructures. This is possible because – unlike in traditional e-cash protocols – the minute nature of payments often allows for increased efficiency under more relaxed guarantees – e.g., upper-capping double-spending instead of full prevention.

In existing micropayment mechanisms, *efficiency* and *correctness* have been two of the main driving design thrusts. Often however micropayment schemes need to also provide *anonymity*, a property that is quintessential for more traditional e-cash but seems harder to achieve here due to efficiency requirements. In e-cash, anonymity is provided by deploying clever yet expensive cryptography or tailored secret splitting. In micropayments however, achieving efficiency, correctness and anonymity at the same time is challenging in no small measure due to the apparently conflicting requirements. For example, often to prevent double-spending (correctness), the identity of payers is included in payments (loss of anonymity). Double-spending could also be prevented by assuming an online bank. The existence of an always available online bank may be however an unreasonable assumption in many distributed scenarios. Yet, while it seems reasonable to sacrifice some accuracy to get efficiency, the price of privacy is inestimable.

Here we introduce a set of efficient, correct and anonymous micropayment mechanisms and proof of concept implementations thereof. Users can make untraceable, anonymous micropayments to each other and several micropayments can be aggregated and cashed once. Illicit behavior such as overspending is detected even when a tunable small amount of cash has been overspent. In such a case only, perpetrator identities are revealed. The mechanisms are practical with minimal overheads and support thousands of transactions per second.

The rest of this paper is structured as follows. We discuss related work in section II and cryptographic tools that are used in section III. The adversary model and formal definitions are provided in section IV. We present the three micropayment mechanisms with full security proofs in section V, VI and VII respectively and followed by performance evaluation in section VIII.

Bogdan Carbunar is with the School of Computing a nd Information Sciences at the Florida International University, Miami, FL. E-mail: carbunar@gmail.com.

Yao chen is with the Computer Science Department at the Stony Brook University, Stony Brook, NY. E-mail: yaochen@cs.stonybrook.edu.

Radu Sion is with the Computer Science Department at the Stony Brook University, Stony Brook, NY. E-mail: sion@cs.stonybrook.edu.

II. Background

Micropayments. In [4], we introduced two micropayment mechanisms – ORPay and PlusPay. In this paper, we extend the work by devising a new protocol, CoinPay, and adding formal definitions and proofs for the three mechanisms. CoinPay provides full anonymity and overspending protection. However, unlike PlusPay, CoinPay does not require the use of the anonymized channel during communication with the bank.

A number of micropayment schemes have been proposed. These include PayWord [5], MicroMint [5], PayTree [6], Peppercorn [7], Millicent [8], Netcard [9], MPTP [10], Lipton and Ostrovsky's coin flipping-based scheme [11], μ-iKP [12], PPay [3], PAR [13]. They are all important work. Due to space limitation, in the following we detail a few of them which are most related to our schemes.

PayWord [5] deploys hash chains for implementing the basic unit payment and only require a signature per session as detailed in section IV-C. Payments can be aggregated. To prevent overspending, the payer identity is included in the payments, thus defeating anonymity. PayTree [6] is building upon PayWord to further reduce the number of required expensive crypto signatures by building a Merkle Tree to efficiently authenticate multiple chains. While two of our solutions may also use the idea of a Merkle tree over identity shares to improve efficiency, we note that neither PayWord nor PayTree attempt to achieve anonymity – this is the subject of our paper. MicroMint [5] coins are hash-colliding values in a model where the bank is assumed to have an advantage in producing hash collisions over other parties. The solution achieves its purpose to eliminate public key operations yet requires the bank to keep track of all coins to prevent double spending and coin forgery. This comes at the expense of practicality and anonymity. Micropayments have also been built on electronic lottery primitives. In Peppercorn [7] "one cent" consists of a lottery ticket with a 1% probability of winning one dollar. This results in a reduction of bank-side overheads at the expense of absolute fairness – payees get paid "on average" and with no anonymity. Specific application-oriented (non-anonymous) schemes have also been proposed. In PPay [3] – targeted at P2P networks – the symmetric nature of the inter-peer relationships (peers can be both payees and payers) is deployed to reduce bank overhead. We refer the interested reader to [14], [15] for a case study and detailed discussions on the costs of micropayments.

E-cash. The use of blind signatures and of the cut-and-choose protocol for providing untraceable electronic cash payments was proposed in [16] [17] [18] [19]. The problem of transferable e-cash was analytically studied first by Chaum and Pedersen [20]. The work of Brands [21] proposes a primitive called *restrictive blind signatures* to replace the high cost of blind signatures that use the cut-and-choose technique. While in our work we have used the latter technique to illustrate our protocol, for real implementations, Brands's solution could be employed. Franklin and Yung [22] propose the use of a trusted entity (trustee) that collaborates with the bank at withdrawal and deposit to provide a computation efficient on-line cash system. Trustees (either on-line or off-line) were proposed to provide variable degrees of anonymity for e-cash [23] [24] [25] [26]. Stadler et al. [23] introduced the notion of coin tracing and introduced several tracing mechanisms, requiring the trustee to be on-line at withdrawal. Camenisch et al. [25], Frankel et al. [26] and Davida et al. [24] proposed payer and coin tracing mechanisms using off-line trustees. In our work however, the payer and payee anonymity is essential and requires the bank to be unable to link the payer and payee even when colluding with one of them. Camenisch et al. [27] propose an efficient off-line anonymous e-cash protocol that offers also user exculpability: the bank can expose double spenders to third parties. In addition, a user can withdraw not one, but 2^l coins, where each coin can be spent unlinkably. The result is interesting in that the storage required for the 2^l coins is only $O(l+k)$, where k is a security parameter. Moreover the solution also allows traceability of coins without a trusted third party. That is, once a user double spends any of its coins, all its spendings, of any of the 2^l coins, can be traced. The coin storage cost of this extension is only $O(lk)$. In [28] Camenisch et al. proposed the notion of endorsed e-coins: a lightweight endorsement x and the rest of the coin which is meaningless without x. Endorsed e-coins allow users to exchange e-cash by exchanging endorsements. Two practical scenarios are studied, an optimistic and unlinkable fair exchange of e-cash for digital goods and services and onion routing with incentives and accountability for the routers. We note that this work discusses a weaker form of anonymity, specifically not considering payees. This is often undesirable e.g. for anonymous services, rendez-vous points [2] etc, where payees need to preserve their privacy. In our work we also consider payee privacy and provide a solution based on the use of anonymous accounts. Furthermore, the endorsed e-coin generation algorithm requires multiple modular exponentiations, being thus several orders of magnitude costlier than our protocols (cryptographic hash evaluations).

A recent result, PAR [13] allows for a certain degree of "Tor"-anonymity for payments in Tor networks under the assumption that " users can only observe the traffic going through them and a limited amount of the rest of the network traffic". Sender anonymity is preserved partly through propagating payments constructed by allowing nodes to only pay their immediate neighbors. The protocol deploys two types of coins: A-coins (anonymous coins) and S-coins (signed coins). S-coins contain identities and can be used by malicious insiders and the bank to compromise anonymity. A-coins are implemented using a variant of traditional e-cash that can often become too expensive (in both cost and execution times) for arbitrary small micropayments. The paper does not evaluate the cost for generating A-coins. Moreover, it is not clear how PAR can be extended to work for arbitrary micropayment scenarios (e.g., payments outside of Tor, where there are no intermediate nodes to preserve sender's anonymity). We believe that micropayments are likely to be deployed in systems requiring high throughputs. In our work we show that expensive modular arithmetic operations for each individual payment coin are unnecessary. Instead, our use of signatures is limited to one per micropayment chain. Moreover, we prevent overspending probabilistically, through the forced revealing of random identity shares during micropayment spend operations.

Ngan et al. [29] introduced an incentive design for Tor. Although it is not a payment protocol, it achieves the purpose of rewarding "good" relays by having the globally trusted directory servers to actively and secretly measure the performance of each Tor router and give high performance routers gold stars on the router list. Traffic initiated from such routers will have higher priority in the Tor network. However, the accuracy of this scheme depends on the frequency of the measurement. As the Tor network grows, this may become a significant overhead for the directory servers.

While a number of the above micropayment mechanisms satisfy some of the correctness and efficiency properties outlined in Section IV-B, in the following we introduce a first set of practical micropayment mechanisms with increasing degrees of anonymity.

III. Preliminaries

We require several cryptographic primitives: (i) a standard cryptographic hash, fast, collision resistant and preimage and second preimage resistant (we use the notation $H(x)$), (ii) a semantically secure [30] encryption scheme, (iii) an unforgeable signature scheme and (iv) a random oracle G.

A. Anonymizers

In our work we use the concept of a network anonymizer, or mix network [2]. Mix networks consist of serially composed servers, each transforming a set of input messages into a permuted and re-encrypted set of output elements. Mix networks satisfy the following properties: (i) operate correctly, that is, the output should correspond to a permutation of the input and (ii) provide privacy, that is, an observer should not be able to determine which input element corresponds to a given output element in any way better than guessing. Chaum [31] was the first to formalize the concept of mix networks, while numerous other works [32], [33], [2], have provided various solutions.

B. Blind Signatures

The purpose of blind signatures is to allow a user to obtain a signature from a signer, such that (i) the signer does not learn information about the signed message and (ii) the user cannot obtain more than l signatures after l runs of the signing protocol. The first property is called *blindness* and the second property is called *unforgeability*. The formal notion of security for blind signatures is security against one more forgery [34], [35]. In the following we assume Chaum's [36] cut and choose blind signature protocol.

C. Threshold Secret Sharing

Our constructions use a threshold secret sharing mechanism based on the original secret sharing problem [37], [38].

Informally, a $(k+1, n)$ threshold secret sharing solution TSS takes as input a value X and generates n "shares" thereof. Any $k+1$ or more shares are enough to re-construct X, however, any k or less shares cannot be used to reconstruct X. A TSS scheme needs to provide a "hiding" property, formalized in the following using an indistinguishability game run between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. Note that most existing TSS solutions are information theoretic secure. In the game, the participants are given access to a $(k+1, n)$ secret sharing oracle SSR, that given a value, produces n shares of the value.
Hiding:

- $\mathcal{C}$ sends the parameters k and n to $\mathcal{A}$.
- $\mathcal{A}$ generates two values R_0 and R_1, selects k indexes $i_i, .., i_k \in_R [1..n]$ and sends them all to $\mathcal{C}$.
- $\mathcal{C}$ selects a bit $b \in_R \{0, 1\}$ uniformly random, invokes SSR to generate shares $sh_{1b}, .., sh_{nb}$ of R_b and sends $sh_{i_1 b}, .., sh_{i_k b}$ to $\mathcal{A}$.
- $\mathcal{A}$ invokes SSR on R_0 and R_1.
- $\mathcal{A}$ outputs its guess b' for the bit b.

We call a TSS solution *semantically secure* if any adversary $\mathcal{A}$ has only negligible advantage in guessing the bit b. That is, Pr$[b' = b] = 1/2 + \epsilon$.

D. Commitment Schemes

A commitment scheme is a triple $(Gen, CMT, Open)$ operating over messages from the space $\mathcal{M}$. Gen generates a public commitment key. CMT takes as input a message $m \in \mathcal{M}$ and produces a commitment value for m. Open takes as input a commitment value and additional information and produces either a message from $\mathcal{M}$ or outputs error. A commitment scheme is correct if $Open(CMT(m)) = m$. A commitment scheme is useful to us if it is *hiding* and *binding*.
Hiding: Informally, it is hard for any PPT adversary $\mathcal{A}$ to generate two messages such that $\mathcal{A}$ can distinguish between their commitments. Specifically, $\mathcal{A}$ produces two messages m_0 and m_1, the challenger $\mathcal{C}$ flips a coin b and sends $\mathcal{A}$ the commitment CMT(m_b). $\mathcal{A}$ has only negligible advantage over 1/2 of guessing b.
Binding: It is hard for any PPT adversary $\mathcal{A}$ to find two messages whose commitments are equal (collision). Specifically, the probability that $\mathcal{A}$ finds a triple (c,m,m'), $m \neq m'$, such that $Open(c) = m$ and $Open(c) = m'$, is negligible.

IV. Model

Let B denote the "bank", any authority that manages payment accounts. It does not need to be centralized and in real deployments this functionality can be delegated to a pre-existing trusted service such as a distributed directory [1]. Let U denote a payer and V be a service payee (e.g., a vendor). B is trusted to correctly withdraw and deposit payments upon valid requests. U and V can be honest or malicious, by all means to break the protocol. Let $Id(X)$ denote the unique identity associated with participant X. Let $\mathcal{U}$ denote the set of *active* payers – payers with open accounts with a positive balance. Let $[U]$ denote the knowledge of a party U, including the bank. $[U]$ consists of all the messages sent and received by U and any knowledge derived from them. We denote with $\{M\}_k$ the encryption of message M with key k. $X \in_R D$ is a random choice of value X from domain D. The notation $Pr_X(Y)$ denotes the probability of event Y given input X.

Adversary. We assume a computationally bounded PPT adversary $\mathcal{A}$ that may collude with or masquerade as any number of vendors, payers and the bank. Specifically, in each security property defined in the following, $\mathcal{A}$ is assumed to control all parties except the party of concern in the property. This adversary is more powerful than the one assumed in Tor – to preserve anonymity, Tor requires at least one non-malicious participant in any circuit of routers. Moreover Tor only guarantees unlinkability of the source and the destination and not full anonymity. It is at least as strong as in related work [13] where "users can only observe the traffic going through them and a limited amount of the rest of the network traffic".

A. Micropayments

Micropayments are financial transactions of small monetary value. An *anonymous micropayment scheme* is a set of protocols μP = {BKGen, UKGen, Withdraw, InitChain, Spend, Deposit, Verify}. We now describe the functionality of each protocol, using a framework similar to the one used by Camenisch et al. [27].

- **BKGen**$(1^k, params)$: Invoked by the bank to generate public and private parameters.
- **UKGen**$(1^k, params)$: Invoked by each user to generate public and private parameters.
- **Withdraw**$(U(pk_B, sk_U, m), B(pk_U, sk_B, m))$: Protocol executed between a user U and the bank B to allow U to withdraw m coins from its account with B. If U's account balance exceeds m, U obtains a payment token P of value m, while the balance of U

[1]In TorPay, the bank is a small component attached to the main Tor Directory service.

with B contains m less coins. Otherwise, Withdraw returns ERROR to both participants.

- **InitChain**$(U(sk_U, P), V(sk_V, pk_B))$: Protocol executed between a payer U and a payee V to allow U to initialize a micropayment chain, given a payment token P obtained during a previous run of Withdraw. The output for U consists of a micropayment chain μCHN or ERROR. The output for V is either a commitment CMT to the micropayment chain or ERROR.
- **Spend**$(U(sk_U, pk_V, \mu CHN, P, l_s), V(sk_V, pk_B, CMT, l_s))$: Protocol run between a payer U and a payee V after the InitChain protocol has completed and l_s micro-coins from μCHN have been spent by U to V. It allows U to spend another micro-coin with V. The output for the payee is one micro-coin and a deposit tuple $D = (PP, w_{l_s})$ or ERROR. The output for the payer is a decremented balance or ERROR.
- **Deposit**$(V(sk_V, pk_B, D), B(pk_V, sk_B))$: Protocol run between a payee V and the bank B and allows V to deposit a set $D = (PP, w, k)$ containing k micro-coins into its bank account, using the last coin w and the proof of payment PP. If w is not valid or PP does not verify, Deposit outputs ERROR. If an overspending is detected (either by V or by another payer that transacted with V), Deposit returns the identity of the overspender, the serial number of the overspent micropayment chain and a proof P. Otherwise, the output for the payee is an account balance increased with k.
- **Verify**(U, SN, P): Any user can run this protocol to verify the overspending proof P against a user U, for a micropayment with serial number SN. If the proof reconstructs the identity of U, $Id(U)$, Verify accepts the proof. Otherwise, it outputs ERROR.

The protocols are used in the following manner. First, each participant runs UKGen (or BKGen by the bank) to generate its private and public parameters. Users can call Withdraw to obtain micropayments from their account with the bank. When a user U needs to run a transaction with a vendor V on amount m, U and V run the InitChain procedure once followed by up to m runs of the Spend procedure. A user may deposit micropayments to its bank account by running the Deposit procedure with the bank. While the Deposit procedure may be called with the m aggregated micro-coins, leading to improved communication efficiency, it can also be called for single micro-coins. Note that a payer can redeem unspent micro-coins using Deposit, just like any other payee. An overspending user is detected during a Deposit operation, when its identity is revealed and a proof of overspending is generated. Any user can run the Verify procedure to verify the overspending accusation.

B. Properties

We now define a set of security properties for anonymous micropayment mechanisms, similar to the definitions proposed in [27].

1) Correctness: If an honest user runs Withdraw with an honest bank, no one will output ERROR. If an honest payer runs InitChain to generate a micropayment chain and then runs Spend with an honest payee, the payee will accept (not output ERROR). If an honest payee runs Deposit using a micropayment received in a previous $Spend$, an honest bank will accept it (not output ERROR).

2) Anonymity: An anonymous micropayment solution should not allow the bank, colluding with any number of users, to link a micropayment to a payer or to link micropayment chains to each other. Formally, there exists no PPT adversary $\mathcal{A}$, controlling the bank and all the other users, that has non-negligible advantage over coin-flip (50%) when playing the following games:

Payment Unlinkability:

- $\mathcal{A}$ generates and sends the bank's public key pk_B to $\mathcal{C}$. $\mathcal{A}$ generates and sends m, the standard currency amount in payments. $\mathcal{C}$ runs UKGen for two users U_0 and U_1 and sends their public keys, to $\mathcal{A}$.
- $\mathcal{C}$ generates payments P_0 and P_1 of m coins each, on behalf of users U_0 and U_1 by running Withdraw with $\mathcal{A}$. $\mathcal{C}$ selects a bit $b \in_R \{0, 1\}$ then runs InitChain followed by up to m runs of Spend with $\mathcal{A}$ for payment P_b.
- $\mathcal{A}$ outputs its guess b' for b.

The advantage of $\mathcal{A}$ in this game is defined as Adv($\mathcal{A}$) = Pr[$b' = b$] $- 1/2$. We say that a micropayment solution provides payment unlinkability if no PPT $\mathcal{A}$ has non-negligible advantage in this game.

Note that this game does not call the deposit function – the adversary already includes the bank. Also, we only consider adversaries that do not ask a user to overspend: Spend is called up to m times for any payment.

Finally, we define the notion of payee anonymity, describing the inability of an adversary to guess the identity of a user performing a deposit operation [39], [40], [41], [42]. We define it in terms of the following game, where the challenger $\mathcal{C}$ controls two users V_1 and V_2 and the adversary $\mathcal{A}$ controls the bank and all the other users.

Payee Anonymity:

- $\mathcal{A}$ generates the public key pk_B of the bank, the public key pk_U for a user U and m, the standard currency amount in payments, and sends them to $\mathcal{C}$. $\mathcal{C}$ runs UKGen for two users V_1 and V_2 and sends their public keys to $\mathcal{A}$.
- $\mathcal{A}$ generates a payment P of m coins, on behalf of U. $\mathcal{C}$ selects a bit $b \in_R \{0, 1\}$. $\mathcal{A}$ runs InitChain followed by up to m runs of Spend for P with $\mathcal{C}$. $\mathcal{C}$ acts as user V_b.
- $\mathcal{C}$ runs Deposit for the m coins received from $\mathcal{A}$, on behalf of V_b.
- $\mathcal{A}$ outputs its guess b' for b.

The advantage of $\mathcal{A}$ in this game is defined as Adv($\mathcal{A}$) = Pr[$b' = b$]$-$ $1/2$. We say that a micropayment solution preserves payee anonymity if no PPT $\mathcal{A}$ has non-negligible advantage in this game.

3) Balance: No coalition of payers and payees should be able to convince the bank to accept a micro-coin that is not valid. We formally define this property in terms of a one-more-forgery game. We expand the definition of the game for our CoinPay and PlusPay solutions, in their respective sections.

4) Culpability: Given a micropayment μCHN of value m, we say a user overspends it if it runs Spend with μCHN more than m times. A user has a small probability of overspending without being detected and without revealing its identity.

5) Exculpability: No coalition of users and the bank can frame another (honest) user for over spending. Formally, an adversary $\mathcal{A}$ controlling the bank, plays the following game with the challenger $\mathcal{C}$ controlling a target user U.

- **Setup**: $\mathcal{A}$ generates the public key of the bank pk_B and sends it to $\mathcal{C}$. $\mathcal{C}$ calls UKGen for a user U and sends pk_U to $\mathcal{A}$. The following step is then executed n times.
- **Query**: $\mathcal{C}$ interacts with $\mathcal{A}$ by calling Withdraw to obtain m coins under serial number SN_i. $\mathcal{C}$ interacts with $\mathcal{A}$ by calling InitChain, then Spend up to m times on the micropayment with serial SN_i.
- **Success Criterion**: $\mathcal{A}$ outputs a serial number SN and proof P.

noindent We define the advantage of $\mathcal{A}$ in this game to be $Adv(\mathcal{A}, n) = Pr[Verify(U, SN, P) = 1]$. We say that a micropayment solution provides exculpability if no PPT $\mathcal{A}$ has non-negligible advantage in this game.

6) Efficiency: Ideally, micropayments should also be efficient and feature the following properties.

Aggregation: Micropayments can be combined into a macro-payment. The macro-payment can be redeemed with the bank for an amount equivalent to the sum of all combined micropayments.

Low overheads: The micropayment transaction protocols have to be computation and communication efficient relatively to their deployment environment.

C. Micropayment Example: Payword

Payment chains were first introduced in Payword [5] by Rivest and Shamir. A payer first contacts the bank to obtain a certificate of credit. It then generates a long one-way hash chain, starting with a random end-element ("the root") which is signed and provided to the payee. Individual chain elements are then used in inverse order (to prevent forgery – due to one-wayness of the hash) as coins. Payees can verify coin validity for each individual payment by re-hashing the received value and comparing it with the previously received payment. The number of so far received coins from a chain represents the value of the total payment. To cash this payment the payee only needs to present to the bank the payer-authenticated root of the hash chain and the last-received chain element. By re-generating the hash chain portion from this value to the root in natural order, any party can verify that the chain is authentic. Basic primitives for Payword [5] are as follows.

- **CertGen**$(U(pk_U, CC, A_U, pk_B), B(sk_B, pk_B))$ U sends its public key pk_U, credit card number CC and address A_U to B. B generates a certificate C_U of format $C_U = \{B, U, A_U, pk_U, E, I_U\}_{sk_B}$, where E is the expiration time of the certificate and I_U is other information, including a certificate serial number, credit limits, bank information.
- **InitChain**$(U(sk_U, C_U), V(sk_V, pk_B))$ Let $w_m \in_R \{0,1\}^k$ be a random number chosen by U. U generates a micropayment chain $\mu CHN = w_0, w_1 ... w_m$, where $w_i = h(w_{i+1})$, for $i = m-1..0$. w_0 is called the root of the chain and $w_1, ... w_m$ are called PayWords. U computes and signs a commitment CMT to the root of the chain and sends the commitment along with its certificate C_U to V. V verifies the signatures. If either verification fails, V generates ERROR.
- **Spend**$(U(sk_U, pk_V, \mu CHN, l_s), V(sk_V, pk_B, CMT, C_U, l_s))$ U sends to V the $l_s + 1$th PayWord, w_{l_s+1}. V verifies that $h(w_{l_s+1}) = w_{l_s}$. If it fails, V generates ERROR. Otherwise, V generates the deposit tuple $D = (CMT, w_{l_s+1})$.
- **Deposit**$(V(sk_V, pk_B, D), B(pk_V, sk_B))$ V sends the deposit set $D = (CMT, w_k, k)$ containing the last micro-coin received, w_k, to B which checks the "well-formed"-ness of CMT and also that $h^k(w_k) = w_0$. If either check fails, the bank generates ERROR. Otherwise, it credits V's account and debits $U's$ account with k.

In the case where a payer U does not spend all the paywords generated in InitChain, it can run the Deposit protocol with the bank and redeem the unspent portion of its chain.

Note that PayWord does not satisfy anonymity (unlinkability and indistinguishability) properties. This is because in InitChain, the payer commits to the root of the micropayment chain by signing it with its private key. Thus, payers can be identified and linked to any generated micropayment as well as to any transaction in which they participate. The payee's identity will also be revealed during the Deposit procedure.

We have summarized the key properties of the existing micropayment schemes in table I.

We will describe the following three protocols below in the context of PayWord. PayWord is a good starting point because it naturally satisfies offline verification and aggregation properties and it has a low overhead: During *Spend* transactions, the bank is offline. Since the spender's identity is included in the payment, overspending is natively discouraged and immediately discovered. Moreover, aggregation is satisfied since payees only need redeem the last micropayment received. Finally, the costs include one signature generation and verification per micropayment chain and one hash computation per micropayment spent.

Protocol	**Anonymity**	**Efficiency**
PayWord, PayTree	No	Hash & amortized signature
MPTP, NetCard	No	Hash & amortized signature
Millicent, MicroMint	No	Hash only
Peppercorn	No	Signature required
Coin flipping	No	Signature & zero knowledge
Par	Partially	Signature required

TABLE I
SUMMARY OF KEY PROPERTIES OF EXISTING MICROPAYMENT SCHEMES.

V. ORPAY: ONION ROUTING PAYMENTS

We start by investigating the use of micropayments in Tor [2] as a means to provide quality of service and motivate system participation. This will constitute a first step to assess their feasibility and efficiency in real deployments. Conceptually, Tor routers will be rewarded with micropayments for correct traffic relaying – these can then be aggregated and deposited into accounts provided through a "banking" service run by Tor's directory. The accounts' balance can then be used as actual cash in webclick-like incentive schemes, in QoS enforcement (e.g., by prioritization of traffic) or in reputation-based mechanisms. For example, routers can specify in their router description that they only accept connections (and traffic) from parties whose balance exceeds a threshold.

We first note that Tor only guarantees unlinkability of the source and the destination but not full anonymity. Moreover, naturally, by its very nature, such an incentive mechanism will not hide identities (of payers or payees). Yet, it is important to at least not compromise these existing k-unlinkability properties. We will achieve this by coupling the fact that routers are simultaneously part of multiple circuits with a design in which routers pay on their own for forwarded traffic. These properties then guarantee the ability to hide traffic origins as well as source/destination associations in the Tor adversarial model.

A. Protocol

Algorithm 1 shows the pseudo-code of ORPay. When a user (the source) needs to send data through Tor, it starts by creating a circuit consisting of L routers (L defaults to 3), by calling the circuitExtend operation (Algorithm 1, lines 5-15). Tor builds circuits incrementally; in a first step the source creates a connection to the first Tor router. Each router extracts the next hop from the packet's circuit (line 6) and extracts the micropayment root commitment (line 7). It then verifies the validity of the commitment (line 8) and drops the circuit if it fails (line 9). If the router is not the last in the circuit (line 11), it runs InitChain with the next hop (line 12) to establish a micropayment chain for future use. It then extends the circuit to the next hop router (line 15), including the commitment of the chain previously generated. This process is performed L times, once for each link in the Tor circuit (the final link is between the last router and the intended destination and there is no payment activity involved).

The actual data transfer, shown in the receiveRelayCell operation (lines 16-27), is executed by each router only when

Algorithm 1 ORPay.

```
1.Object implementation ORouter;
2. int coinsNextOR;      #coin count owed next OR
3. int L;      #circuit length;
4. int pos;      #position in circuit;
5. Operation circuitExt(Cell cell, Circuit circuit)
6.     ORouter nextOR = circuit.getNextOR();
7.     CMT = removeCMT(cell);
8.     if (valid(CMT) == false) then
9.         return ERROR;
10.    fi
11.    if (lastRouter() == false) then
12.        CMTNext = initChain(nextOR);
13.        cell.append(CMTNext);
14.    fi
15.    nextOR.circuitExtend(cell, circuit);
16. Operation receiveRelayCell(Cell cell,
        Circuit circuit, int cellDir, intn)
17.    ORouter nextOR = circuit.getNextOR();
18.    #Only outward packets contain payments.
19.    if (cell.command == CELL_RELAY &&
            cellDir == OUT) then
20.        Payment coins = getPayment(cell);
21.        if(!valid(coins)||coins.val! = L − n + 1) then
22.            return ERROR;
23.        Payment coinsNext = newPayment(L − n);
24.        Spend(nextOR, coinsNext);
25.        cell.insertPayment(coinsNext);
26.    fi
27.    circuit.append(cell, cellDir);
```

received packets can be decrypted with the router's private key. In receiveRelayCell, *conceptually* the source will include L micropayments in each packet it sends to the first router – for this the source and the first router run the $Spend$ protocol. Recall that the L micropayments are part of the chain initialized during the circuit establishment step. Without loss of generality we also assume that forwarding a *Relay* packet[2] is worth one micropayment. Each router extracts the next hop from the circuit (line 17) and verifies that the packet is of type RELAY and outbound (the only packets that contain payments). If this is the case (line 19) the router extracts its payment from the packet (line 20). It verifies its validity and the fact that it contains L-n+1 coins (line 21), where n is the router's position in the circuit. It then generates $L - n$ coins (line 23) and spends them with the next hop router (line 24).

Routers can aggregate micropayments and report them to the bank at their leisure. The bank updates router ranks periodically by calculating the performance of each router, for instance as the ratio of micropayments earned to micropayments spent. Although each router holds an account, there is need to worry about overspending or double spending. Selfish routers which use Tor only to relay their traffic but not provide service to others people will end up with very low ranks.

Often the destination host can generate (significant) traffic back to the source. Even though initiated by the destination, the source might be the one that is expected to pay for it. Similar to uploading, every router need to pay $L - n$ coins to its successor, so that everyone gets one coin for every packet. This can be done retroactively by having the routers piggybacking micropayments to future outward packets. Note that the source can also pay ahead for traffic initiated by the destination: during the circuit initialization step, the source provides payment for the first packet expected to be sent by the destination and similarly, outward packets will contain micropayments for future packets.

In practice, numerous optimizations can be deployed to the above protocol. For example, a single payment token can be included for multiple packets. Also, to accelerate the protocol, a sliding window scheme can be used to allow the destination to send several packets at a time. If the source trusts the destination to correctly acknowledge receipt of packets, the potential cash loss due to unfair behavior can be bounded by the size of the sliding window W. The upper bound on cash loss is $W * e * L$, where e is the value of each payment amount and L is the number of routers.

The benefit that the payment scheme brings is clear: the more traffic a Tor router relays for others, the higher rank it will get. As a result, its personal traffic will be preferred in the Tor network. Note that pure Tor clients (not routers) are not given rank and have the lowest priority in the Tor network.

B. *Implementation: ORPay*

We implemented ORPay, a proof of concept prototype of the above mechanisms. ORPay deploys out of band (OOB) communication for payment primitives and control messaging. The "Bank" is implemented in C (using OpenSSL for cryptography) as a stand-alone component attached to the Tor directory server. One of the main raison d'etre of ORPay was to evaluate the practicality of "payment chain" based micropayment approaches. We thus ran a number of experiments to evaluate the associated overheads. The controlled environment consisted of a set of interconnected physical machines (with 1.66GHz Intel Core Duo CPU and 2 GB RAM) running one directory server and a set of tor routers based on VMs, each router in turn running Tor with default settings under Ubuntu Linux. The average observed inter-client bandwidth was 500-600KB/s, the average latency between physical machines was 1-2ms (0.5ms inter-VMs on the same machine). ORPay was set up to send one micropayment for every 20 routed packets.

In a first experiment we evaluated the per-hop latency overheads introduced by ORPay. These latency overheads were mainly a result of host-side payment processing as well as payment propagation network latencies. The payment processing does not contain any expensive public key operations (the signature cost in InitChain step only happens once per session and the cost is amortized). The out of band nature of the design resulted in values of about 0.9ms per 3 relay setups, averaging under 300 microseconds per relay.

Next we aimed to understand the impact of the micropayment mechanism on core throughput. We benchmarked a number of file transfers of increasing amounts of data. As payments average around 20 bytes and the standard Tor frames are 512 bytes, a general worst-case upper bound of just under 4% on bandwidth overhead can be established (for one

[2]There are two types of packets in Tor, *Control* and *Relay* packets. *Control* packets which contain circuit building and destroying commands are not considered for payment. *Relay* packets carry end-to-end data, and they are what the source needs to pay for.

payment token per frame). The observed overheads averaged under 2%, due to multiple payload frames per token.

Collected payments can be deposited in the bank during network idle time. The overhead for the directory server to process one deposit consists of reading data (a payer signed commitment CMT and the last payword) from the connection, one signature verification and a number of cryptographic hashes. For a payment chain of length 1,000, the observed overhead was under 2ms. As discussed, by its very nature, the above reputation/incentive mechanism will not hide payers or payees identities. This solution does not provide payment unlinkability, payment indistinguishability, or payee anonymity. However, by letting each router only pay its successor, and considering the fact that each router can be simultaneously part of multiple circuits, it hides the traffic origins as well as source/destination associations [3].

VI. CoinPay: Overspending Protection

CoinPay is the next level protocol which aims to provide protection against overspending. At an overview level the solution proceeds as follows. Payers start by explicitly withdrawing cash from their accounts before being able to generate micropayments. Additionally, instead of directly signing micropayment chains, and thus revealing their identity, payers will ask the bank to partially blindly sign the w_0 roots of the micropayment chains in a standard cut and choose protocol which also allows the bank to verify the generated payment's structure. This ensures that (i) payers are generating a valid micropayment chain, thus allowing the bank to withdraw the correct amount from their accounts, and (ii) the bank cannot link the chain to honest (not overspending) payers.

To prevent a payer from overspending, threshold splitting is employed to generate shares of the payer's identity. These "identity shares" are then directly linked to micropayments: for a micropayment chain of value m, $n > m$ identity shares are generated, such that any $m+1$ shares are enough to recover the payer's identity. In every micropayment, the payer is forced to reveal a randomly chosen identity share to the payee. When the number of revealed shares exceeds m (in the case of overspending), the bank will be able to identify the over spender. Additionally, to prevent payers from providing fake shares to payees, during payment generation, the bank signs not only the w_0 micropayment chain root but also the payer's commitments to the identity shares. Later, upon spending the micropayment, the payer needs to reveal the identity share and open the commitment to prove this share was indeed part of the set of valid identity shares for the current payment.

Of concern given the cut-and-choose approach of generating micropayments is the probability to cheat. Even if the probability is small, if successful, a cheating payer can end up with a very large payment. We address this issue by allowing only predefined micropayment chain values (e.g., 1$, 5$, 10$). The bank will not accept micropayment chains of a different value. The upper bound on the chain value can be defined to be a function of the payer's probability to cheat. Moreover, the bank can punish detected cheating attempts for instance by penalizing the culprit's account with a function of the cheated amount. Given the small probability of a successful attack, such a strategy can be a powerful deterrent.

A. Solution

We now describe CoinPay's protocols.

Withdraw$(U(pk_B, sk_U, m), B(sk_B, m))$**.**

U runs Chaum's partially blind signature protocol [36] with B by building equivalent payment messages. For *each* payment message, U performs the following steps

- Generate $\{sh_1, .., sh_n\}$, shares of Id(U), where $n > m$ [4] such that any but no less than $m+1$ shares can be used to reconstruct $Id(U)$.
- Generate n "identity shares" $IdShare_i = \{sh_i, i, m\}$, where i is a unique sequence number. i is used later to prove that the share is not a duplicate. Commit to the value of the identity shares. Let C = π $\{$ CMT($IdShare_1$),..,CMT($IdShare_n$)$\}$, where π is a random permutation.
- Construct micropayment chain of length m and w_0 as root.
- Generate the payment message $M = \{m, SN, w_0, C\}$, where SN is a unique serial number.

Note that w_0 and C are different for each of the payment messages. U blinds the payment messages and sends them to B. B request U to unblind all but one payment message. For each unblinded message B verifies that:

- The first field of the unblinded payment message is m.
- Each identity share has a unique sequence number and its last field is m.
- Any $m+1$ identity shares correctly reconstruct Id(U).
- The n commitments to identity shares are correct.

If any of these checks is not satisfied, the bank generates ERROR. Otherwise, it withdraws m currency units from U's account and sends the signed unrevealed blinded message to U – who is then able to recover the anonymous micropayment chain $P = \{m, SN, w_0, C\}_{sk_B}$.

InitChain$(U(sk_U, P), V(sk_V, pk_B))$**.** InitChain inherits and extends the behavior of Payword's InitChain procedure. In addition, in CoinPay's InitChain, U sends $P = \{m, SN, w_0, C\}_{sk_B}$ to V. V verifies B's signature and the fact that the first field of the signed message is m. If any of these checks fails, V returns ERROR. Otherwise, V stores P.

Spend$(U(sk_U, pk_V, \mu CHN, P, l), V(sk_V, pk_B, P, l))$**.** V generates a random number R_V and sends it to U. U performs the following steps:

- Generate the next coin in the micropayment chain, w_l. Send w_l to V.
- Use Id(V), V's random value R_V, the root of the micropayment chain and a sequence number as input to the random oracle G and select the index of one of the n identity shares: $i = G(Id(V), R_V, w_0, l) \ mod \ n$. Send $IdShare_i$ to V along with additional information allowing the verification of the commitment.

[3]This is no longer true for other Internet services such as in cloud computing or even p2p file sharing, where there are no intermediate nodes to hide the sender's identity.

[4]There is a relationship between n and m that will be defined later, e.g., in the proof of Theorem 3.

Let S be the set of identity shares already received by V. Upon receipt of the above values, V performs as follows:

- If $IdShare_i \in S$ request a new identity share. To avoid misunderstanding, both U and V can maintain the list of identity shares consumed so far.
- Verify the validity of the micropayment against the root of the micropayment chain, $h^l(w_l) = w_0$.
- Verify the correct computation of the index of the revealed share ($i = G(Id(V), R_V, w_0, l) \ mod \ n$). Verify the format of the revealed identity share: $IdShare_i = \{sh_i, i, m\}$.
- Verify the validity of CMT($IdShare_i$), using the revealed identity share and the set C.

If any of these verifications fails, V generates ERROR. Otherwise, it adds $IdShare_i$ to the set S. Let $D = (P, S, w_f, w_l, f, l)$ be the deposit set. w_f denotes the first micro-coin (of index f) and w_l is the last micro-coin (of index l) received by V.

Deposit$(V(sk_V, pk_B, D), B(pk_V, sk_B))$**.** V performs the following steps:

- Deposit $D = (P, S, w_f, w_l, f, l)$ to B. $P = \{m, SN, w_0, C\}_{sk_B}$, S is the set of shares corresponding to the deposited micro-coins, w_f and w_l are the first and last micro-coins (of index f and l respectively) from the chain.

For each serial number SN seen so far, B stores a record of format $Rec_{SN} = \{P, IdShare_1, .., IdShare_r, C\}$, where $IdShare_1$, $..IdShare_r$ are shares deposited so far from the corresponding micropayment. When B executes a Deposit, it performs the following actions:

- Verify its own signature on the P value.
- For $i = f..l$, verify that $h^i(w_i) = w_0$.
- Retrieve from local storage the record Rec_{SN} whose first field is the value P. Rec_{SN} may be undefined. Verify that each received identity share is unique. Verify that the commitment of each received identity share is indeed part of the set C (part of the P). Let l $\leq$ k be the number of identity shares that verify and that are not already stored in Rec_{SN}.
- If any of these checks fails, generate ERROR.
- Otherwise, credit V's account with l coins and store all the received identity shares under Rec_{SN}.
- If overspending is detected, that is the number of shares in Rec_{SN} exceeds $m + 1$, recover Id(U) using the shares and publish the proof P={Id(U),Rec_{SN}}.

Verify(U, SN, Rec_{SN}**))** As defined above, $Rec_{SN} = \{P, IdShare_i, i = [1..r], C\}$. To verify overspending charges, perform the following steps:

- Verify B's signature on the P value and the validity of the included identity shares.
- Use the identity shares to reconstruct the identity of the over spender. If the reconstruction fails or its output differs from Id(U), output ERROR. Otherwise accept.

B. Analysis

Correctness: By construction, it is straightforward to see that if an honest user runs Withdraw with an honest bank, the bank's verification step of any $t - 1$ payments will succeed. Thus, no participant will output ERROR. Similarly, if an honest payer runs InitChain and Spend (for the same payment) with an honest payee, the payee's verifications will succeed. Finally, if an honest user runs Deposit for a payment previously received, with an honest bank, the bank's verifications will succeed. Moreover, since the payment was received from an honest user, no over spending will be detected.

Anonymity: The following theorems prove the unlinkability and indistinguishability properties of CoinPay.

Theorem 1: Payments in CoinPay are unlinkable.

Proof: Our proof is based on a reduction from the hiding property of a TSS. Specifically, we assume an algorithm $\mathcal{B}$ that has advantage ϵ_{CP} when playing the unlinkability game. We then build an adversary $\mathcal{A}$ that uses $\mathcal{B}$ as a black box to gain advantage ϵ_{TSS} when playing the hiding game of the threshold secret sharing scheme.

The reduction works as follows. $\mathcal{C}$ sends parameters k and n to $\mathcal{A}$. $\mathcal{A}$ then generates two random numbers, R_0 and R_1, and selects k index values. For simplicity of exposition, let these indexes be $1, .., k$. $\mathcal{A}$ sends the indexes, along with R_0 and R_1 to $\mathcal{C}$. $\mathcal{C}$ selects $b \in_R \{0, 1\}$, generates shares $sh_{1b}, .., sh_{kb}$ of R_b and sends them to $\mathcal{A}$. $\mathcal{A}$ calls UKGen to generate two users U_0 and U_1, such that $Id(U_0) = R_0$ and $Id(U_1) = R_1$. $\mathcal{A}$ then initializes algorithm $\mathcal{B}$ and gives it the public keys of U_0 and U_1, along with k as the number of coins in a payment and n as the total number of identity shares. $\mathcal{B}$, following the unlinkability game, calls BKGen and sends the bank's public key to $\mathcal{A}$. $\mathcal{A}$ runs Withdraw with $\mathcal{B}$ with one modification: since $\mathcal{A}$ only possesses k out of n shares, it fabricates additional shares $IdShare_i = \{sh_{i \mod k}, i, m\}$. In Withdraw $\mathcal{B}$ receives commitments of the fabricated identity shares. During the verification step of Withdraw, if $\mathcal{B}$ requests $\mathcal{A}$ to reveal exactly the $sh_{1b}, .., sh_{kb}$ received from $\mathcal{C}$, $\mathcal{A}$ simply aborts and then repeats. $\mathcal{A}$ then runs the InitChain protocol once and Spend protocol k times with $\mathcal{B}$. After the jth run of the Spend protocol, w.l.o.g., let $R = \{sh_{i_1}, .., sh_{i_j}\}$ be the shares revealed by $\mathcal{A}$ to $\mathcal{B}$. During the $j+1$st run of Spend, $\mathcal{A}$ generates a verifiable index and picks the share corresponding to the index. If the share is not in R, add the share to R and continue the Spend protocol as defined in CoinPay. Otherwise, abort Spend and repeat. After receiving k valid coins, $\mathcal{B}$ is able to output and send to $\mathcal{A}$ its guess b' for the bit b. $\mathcal{A}$ sends b' to $\mathcal{C}$. We now prove the following lemma.

Lemma 1: $\mathcal{A}$ terminates in expected polynomial time.

Proof: $\mathcal{A}$'s interaction with $\mathcal{C}$ requires a constant amount of computation and communication. $\mathcal{A}$'s probability to abort the Withdraw operation is $1 - 1/t$, where t is the number of messages used in the blind signature protocol. Thus, the expected number of calls to Withdraw is t. The complexity of Withdraw is linear in t and n. $\mathcal{A}$ also runs Spend k times. For the $j+1$st run, $j+1 \leq k$, the probability of selecting an index whose share has not yet been revealed is $(n-j \times n/k)/(n-1)$. Thus, the expected total number of calls for Spend is

$$\sum_{j=1}^{k} \frac{n-j+1}{n-(j-1) \times n/k} = \frac{k}{n} \sum_{j=1}^{k} \frac{n-j+1}{k-j+1} \leq \frac{k}{n} n \sum_{j=1}^{k} \frac{1}{j} \approx k \ln k$$

Since Spend's computation and communication is constant, this implies that $\mathcal{A}$ is PPT. ■

When $\mathcal{A}$ terminates, $\mathcal{B}$ has one payment message (one micropayment) and k valid spent coins from it. If $\mathcal{B}$ succeeds in guessing b, that is, the identity of the user that generated the payment, then $\mathcal{A}$ can also guess to which user the shares from C belong to. $\mathcal{A}$'s advantage in the TSS Hiding game equals $\mathcal{B}$'s advantage in the CoinPay Unlinkability game: $\epsilon_{TSS} = \epsilon_{CP}$. ∎

Theorem 2: CoinPay payments are indistinguishable.

Proof: We prove by reduction from the hiding property of TSS. We assume an algorithm $\mathcal{B}$ that has an advantage when playing the indistinguishability game. We then build an adversary $\mathcal{A}$ that uses $\mathcal{B}$ as a black box to gain the same advantage when playing the hiding game of the threshold secret sharing scheme. First, $\mathcal{C}$ sends parameters k and n to $\mathcal{A}$. $\mathcal{A}$ generates two random numbers, R_0 and R_1, then selects indexes $1, .., k$. $\mathcal{A}$ sends the indexes, along with R_0 and R_1 to $\mathcal{C}$. $\mathcal{C}$ selects $b_1, b_2 \in_R \{0, 1\}$, generates shares $sh_{1b_1}, .., sh_{kb_1}$ of R_{b_1} and shares $sh_{1b_2}, .., sh_{kb_2}$ of R_{b_2} and sends them to $\mathcal{A}$. $\mathcal{A}$ calls UKGen to generate two users U_0 and U_1, such that $Id(U_0) = R_0$ and $Id(U_1) = R_1$. $\mathcal{A}$ initializes the algorithm $\mathcal{B}$ and gives it the public keys of U_0 and U_1, along with k as the number of coins in a payment and n as the total number of identity shares. $\mathcal{B}$, following the indistinguishability game, calls BKGen and sends the bank's public key to $\mathcal{A}$. $\mathcal{A}$ runs Withdraw with $\mathcal{B}$ twice, once for each set of identity shares. The IdShare values are computed as in the previous proof – $\mathcal{B}$ receives n commitments of identity shares fabricated from the k shares $\mathcal{A}$ has received from $\mathcal{C}$. $\mathcal{A}$ has two signed micropayment chains. For each micropayment chain, $\mathcal{A}$ runs the InitChain protocol once and the Spend protocol k times with $\mathcal{B}$. the Spend protocol is executed similarly to the previous proof: $\mathcal{A}$ completes each call only if requested to reveal a share not already spent. Similar to the reasoning in the previous proof, it can be seen that $\mathcal{A}$ succeeds to generate and spend two valid micropayment chains in polynomial (in t and n) time. The two micropayment chains have been generated on behalf of two valid users, U_0 and/or U_1. If $\mathcal{B}$ has advantage ϵ_{CP} to guess whether the payments were generated for the same user, then $\mathcal{A}$ has the same advantage in guessing whether the two sets of shares sent by $\mathcal{C}$ reconstruct the same value. ∎

Balance: We define this property in terms of a one-more-forgery game. Specifically, an adversary $\mathcal{A}$ runs Withdraw l times with the bank B. Let S_j be the set of identity shares generated during the jth run of Withdraw, $1 \leq j \leq l$. Let $S = \{S_j | 1 \leq j \leq l\}$. Let P_j be the payment generated in the jth run of Withdraw. $\mathcal{A}$ then outputs a deposit tuple $D = (P, IdShare, w)$ such that IdShare $\notin S$. The advantage of $\mathcal{A}$ is defined to be $Adv(\mathcal{A}) = Pr[Deposit(\mathcal{A}(params_{\mathcal{A}}, D), B(params_B)) = 1]$. CoinPay is said to have the Balance property if no PPT has non-negligible advantage in this game.

CoinPay provides the Balance property. Consider an adversary $\mathcal{A}$ that has a non-negligible advantage in the above game. $\mathcal{A}$ is then able to generate a deposit tuple $D = (P, IdShare, w)$ such that IdShare $\notin S$ and the Deposit procedure succeeds with non-negligible probability. Then, $\mathcal{A}$ is able to either (i) forge a value $P = \{m, SN, w_0, C\}_{sk_B}$ or (ii) produce a value IdShare $\notin S$ such that $CMT(IdShare) \in C$, where C is the set of identity share commitments. Case (i) cannot occur without $\mathcal{A}$ having a non-negligible advantage in forging B's signature. Case (ii) can only occur if $\mathcal{A}$ has non-negligible advantage in the partially blind signature protocol or if $\mathcal{A}$ has non-negligible advantage against the binding property of the commitments scheme.

Culpability: Overspending is prevented through the use of the identity shares. A payer that spends more than m micropayments from a chain of value m, also reveals more than m identity shares, which are then enough to expose its identity. Note that the payer cannot control which share it has to reveal, due to the payee's involvement in the choice (the random R_V). A certain probability exists that in the random choice, a previously seen identity share is selected – which could allow overspending. This attack cannot occur when the payer spends a micropayment chain with a single payee – the payee detects double spending of shares. Instead it can occur when a payer attempts to spend the same micropayment chain with multiple payees. Note that this is not recommended even if overspending does not occur: the bank will detect that the same micropayment chain has been spent multiple times and the payment indistinguishability property is compromised. However, the following theorem shows that even if this attack is launched, its chance of success can be controlled. Let $f = n/m > 1$ be the *overspending control factor* and let the gain of overspending be the number of already spent coins the payer can spend without being detected.

Theorem 3: The gain of a payer when attempting to overspend a micropayment chain with any payee is 1/(f-1).

Proof: Let us assume a payer U that has already spent m out of its total n identity shares and m+1 identity shares are sufficient to reconstruct its identity. U then initiates a new transaction with a vendor V consisting of one execution of InitChain, followed by executions of Spend in which the share to be revealed is one of the m already spent. The expected number of runs of Spend before an m+1st share has to be revealed is $E[m+1] = 1/p_{m+1}$, where $p_{m+1} = (n-m)/n$ is the probability of selecting a new share. Thus, $E[m+1] = n/(n-m)$. The gain of U in this attack is defined as

$$Gain(U) = E[m+1] - 1 = \frac{m}{n-m} = \frac{1}{f-1},$$

since one of the runs of Spend will result in the m+1st share being revealed. ∎

For instance, for $f = 10$, Gain(U) = 11%. Thus, the chance of an attacker of succeeding in this attack is far below the chance of failing. Since the cost of failing (identity revelation, tearing down established Tor circuits, etc) also exceeds the benefit of succeeding, overall, our solution encourages rational users to be honest.

Exculpability: Let us assume that an algorithm $\mathcal{B}$ exists that has non-negligible probability of succeeding in the exculpability game defined in Section IV-B. Then, we can build an adversary $\mathcal{A}$ that has non-negligible advantage in the hiding game of a TSS. $\mathcal{A}$ interacts with the challenger $\mathcal{C}$ in the hiding game, to generate values R_0 and R_1 and to obtain two sets of shares, each of either R_0 or R_1. $\mathcal{A}$ uses each set of shares in the Query step of the exculpability game with $\mathcal{B}$, to generate and spend coins. The coins are generated from user accounts with ids R_0 or R_1. If a Query step fails (see above proofs), $\mathcal{A}$ repeats it (a polynomial number of times in t and m). Let ϵ_S be the advantage of $\mathcal{B}$ in this game, $\epsilon_S = Adv(\mathcal{B}, 2)$ (see Exculpability definition). ϵ_S is the probability that $\mathcal{B}$ produces a serial number SN and a proof P such that Verify(R_b,SN,P) accepts, for $b \in \{0, 1\}$. Then, with probability ϵ_S, $\mathcal{A}$ returns 0 to $\mathcal{C}$ – its guess is that the two sets of shares were for the same number (either R_0 or R_1).

Other Properties: CoinPay naturally provides offline verification and aggregation. Moreover, as our experiments report in Section VIII show, CoinPay's overheads are low.

VII. PlusPay: Payee Anonymity, Efficient Bank

One of the problems of CoinPay is the payer's dependence on the bank to sign each micropayment chain. While this may be reasonable for long chains, it makes little sense for small chains, due to the small return on the generation cost. Yet, small chains are more likely to occur in practice, e.g., in short interaction between payers and payees. Moreover, as mentioned before, a payer cannot use the same micropayment chain with multiple payees, without compromising the payment indistinguishability property. Even if the payer generates a batch of micropayment chains at a time, each payment instance needs to be separately signed (blindly) by the bank. Another problem is that, since each coin is bound to an identity, CoinPay works only for non-transferable coins.

In the following we introduce PlusPay, a protocol which, while not necessarily being more overall-efficient – does not require an online bank at client transaction time. Clients will be free to generate chains even if the bank is not within reach. This is an important efficiency/availability advantage. PlusPay uses a level of indirection: Instead of withdrawing micropayments from its (authenticated) bank account, the payer opens an anonymous accounts with the bank. Each anonymous account has a public/private key pair which may be used by the account's owner to sign micropayments. PlusPay achieves this without allowing the bank to link payers to their anonymous accounts or associated key pairs. The use of anonymous accounts provides an additional property. It hides not only the identity of the payer – while subsequently unlinking it from its payments – but it also allows the payee (vendor) to preserve its anonymity when depositing earned payments. This property can be essential from the perspective of hidden (anonymous) services [2].

A. Overview

Overall, PlusPay works as follows. A payer withdraws e-cash from its bank account. Then, by interacting with the bank through an anonymizer, it opens an anonymous account in which it deposits the previously acquired (un-traceable) e-cash. The anonymizer provides unlinkability between the payer's identity and its anonymous account. The anonymous account is then associated with a public/private key pair, generated by the bank and known thereafter only by the payer that opened it. To commit to a micropayment chain root w_0 – instead of requiring the bank's signature as in the CoinPay solution – the payer will sign it using the private key associated with its anonymous account. Note that in the CoinPay solution, micropayments are linked to an e-cash bill withdrawn from the payer's bank account. Here micropayment chains are similar to checks drawn from a bank-hosted anonymous account. More formally, PlusPay, a micropayment system with payer independence is a set of protocols, PlusPay = {BKGen, UKGen, Withdraw, OpenAC, SplitId, InitChain, Spend, Deposit, Verify}. The functionality of most of these protocols is inherited from the anonymous micropayment scheme described in Section IV-A. We now describe the functionality of the new logic.

B. Solution

Our solution relies on the existence of a mix network, denoted by AChan (see Section III for definition and properties).

Withdraw$(U(pk_B, sk_U, m), B(pk_U, sk_B, m))$. U generates a payment of format (SN, m) and a token of format $(token, m)$, where SN (serial number) and token are independent random numbers (using different formats to be distinguished). U asks B to partially blindly sign the payment and token, while also allowing B to verify their correctness: the format of SN and token and the value of m. If the verification fails, B generates ERROR. Otherwise, U obtains a signed anonymous e-cash bill, $EC = \{SN, m\}_{sk_B}$, and a token $TK = \{token, m\}_{sk_B}$ with SN and token unknown to B.

OpenAC$(U(pk_B, sk_U, EC, m), B(sk_B, m))$. U performs the following steps:

- Generate a new public/private key pair (pk_{AC}, sk_{AC}) for a new anonymous account AC.
- Send pk_{AC} and the blindly bank-signed e-cash EC obtained during the Withdraw step, to B, over AChan.

When B receives this message it performs the following:

- Check the validity of the e-cash (the signature and whether it has been spent before). If EC is invalid, generate ERROR.
- Open an anonymous account AC identified by a (random) serial number SN_{AC} and initialize it with the m-valued currency of EC.
- Sign balance certificate BalCert(AC) = $\{pk_{AC}, SN_{AC}, m\}_{sk_B}$, and send the anonymous account information AC=$\{SN_{AC}, m, BalCert\}$ to U over AChan.

We note that this is the only step requiring an anonymizer in our protocol. Its associated traffic is negligible. When deployed for micropayments in anonymizers, the system can be set up to allow new payers to use the anonymizer for free to open their account. This avoids the circularity of new payers being unable to (micro)pay for anonymizer traffic when joining.

SplitId$(U(pk_B, sk_U, AC, TK, m), B(sk_B, m))$. U performs the following steps:

- Use a $(m + 1, n)$ TSS scheme to split Id(U) into shares $sh_1, .., sh_n$. Generate random number $R_U \in_R \{0, 1\}^k$. Use each share to build an "identity share" of format $IdShare_i = \{sh_i, i, m, \{CMT(SN_{AC})\}_{sk_{AC}}\}$
- Generate the set of commitments of identity shares, C = π { CMT($IdShare_1$),..,CMT($IdShare_n$)}, for a random permutation π.
- Send the bank-signed token TK to B.

- Engage in partially blind signature protocol with B to sign the tuple $\{m, C\}$. At the end of this protocol, B will return $P = \{m, C\}_{sk_B}$ and be assured w.h.p. (a function of deployed blind signature protocol, e.g., cut-and-choose) that:
- Signature on TK verifies.
- TK ensures the existence of an account with balance m.
- TK has not been used before (bank keeps track of used TK values).
- The identity share commitments are correct.

U's outcome, if B does not output ERROR, is a payment of format $P = \{m, C\}_{sk_B}$

InitChain$(U(sk_U, BalCert(AC), P),\ \ V(sk_V, pk_B))$. Besides the inherited behavior of the InitChain procedure of Payword, PlusPay's InitChain consists of the following actions. U generates a micropayment hash chain and commits to its root. Instead of the commitment being generated using U's private key, it is generated using the secret key associated with the anonymous account AC. The commitment has format $CMT = \{w_0, SN_{AC}\}_{sk_{AC}}$. U sends the commitment CMT, the BalCert(AC) certificate and the P value to V. V performs the following actions:

- Verify B's signature on P and that the value of m in P matches the value of m in BalCert.
- Validate CMT by checking that (i) the public key pk_{AC} contained in BalCert can verify the signature on CMT and (ii) the account number SN_{AC} contained in CMT is consistent with the one in BalCert. If any check fails, output ERROR.

Spend$(U(sk_U, pk_V, \mu CHN, P, l), V(sk_V, pk_B, CMT, l))$. V sends a random value R_V to U. U performs as follows:

- Send V a new micropayment coin, part of the μCHN chain (see Section IV-C).
- Send V a provably random selected identity share using the technique described in CoinPay. Let the chosen share be $IdShare_i = \{sh_i, i, m, \{CMT(SN_{AC})\}_{sk_{AC}}\}$.
- Open the commitment CMT to V, revealing SN_{AC}.

Let S be the set of identity shares previously revealed by U to V. V verifies the newly revealed identity share before accepting the micropayment:

- The commitment of $IdShare_i$ is contained in the commitment set C, signed by B (part of P).
- $IdShare_i$ has the expected index $i = G(Id(V), R_V, w_0, l)\ mod\ n$ (as in the CoinPay solution).
- The balance m from $IdShare_i$ matches the ones in BalCert and P.
- The commitment on SN_{AC} is correct.

If any check fails, V outputs ERROR. Otherwise, it adds $IdShare_i$ to the set S. Let the deposit tuple $D = (P, CMT, BalCert, S, R_U, w_f, w_l, f, l)$, where w_f and w_l are the first and last micro-coin received by V, of index f and l.

Deposit$(V(sk_V, pk_B, D), B(pk_V, sk_B))$. V sends to B, over AChan, the deposit tuple D containing micro-coins from index f to l in a chain: the commitment set signed by B (P), the root of the chain signed with the private key of an anonymous account (CMT), the BalCert value, the obfuscating factor R_U and $l - f + 1$ unique IdShare values. B verifies that

- The P and BalCert values are signed with its public key. The CMT value is signed with the private key of the anonymous account whose serial number is contained in both P and BalCert.
- All the identity shares are unique, signed and associated with the same anonymous account SN_{AC}. Moreover, their commitments are included in the commitment set contained in P.
- For $i = f..l$, $h^i(w_i) = w_0$: the $l - f + 1$ micro-coins verify the link to the micropayment chain root w_0 contained in CMT. Moreover, the account balance, m (contained in BalCert), exceeds or equals $l - f + 1$.

If any verification fails, B generates ERROR. Otherwise, it records the shares associated with the serial number SN_{AC} into a record of format $Rec_{SN_{AC}} = \{P, IdShare_1, .., IdShare_r, C\}$, where $IdShare_1$, $..IdShare_r$ are shares deposited so far from the corresponding micropayment. To reduce storage cost and the time required to detect overspending, expired micropayments can be garbage collected and payees will need to cash payments before their expiration date. If more than m shares are collected, B recovers Id(U) using the shares and publishes the proof P={Id(U),Rec_{SN}}. B blindly signs (over AChan) a payment token of value $l - f + 1$. V later provides this payment token over an authenticated channel to B, to deposit the $l - f + 1$ micro-coins into its account.

Note that a payer can call Deposit to redeem unspent micropayments. Its identity will be protected as long as it does not over spend.

Verify(U, SN, Rec_{SN}**)** As defined above, $Rec_{SN} = \{P, IdShare_i, i = [1..r], C\}$. To verify overspending charges, perform the following steps:

- Verify B's signature on the P value and the validity of the included identity shares.
- Use the identity shares to reconstruct and the identity of the over spender. If the reconstruction fails or its output differs from Id(U), output ERROR. Otherwise accept.

Timing of Protocol Calls. Micropayment generation consists of calls to Withdraw, OpenAC and SplitId, in this order. Withdraw and SplitId are performed over an authenticated channel, whereas OpenAC is performed over an anonymous channel. Care must be taken to ensure the bank cannot link these calls through their timing, since otherwise the payer's identity (or its identity shares) could be linked to its anonymous account. While we believe that this is arguably an application specific challenge related to the use of anonymizers in general, nevertheless, in the following, we propose a solution that addresses this issue probabilistically. We illustrate for the account and payment setup phase, but note that *the same mechanisms can be deployed in any and all other phases* as required by the application specifics. The bank divides time into epochs, whose start times and durations are made public. W.l.o.g., let us assume that all epochs have the same length, T_e (used below). Each epoch is divided into three frames: Withdraw, OpenAC and SplitId. The bank will answer Withdraw, OpenAC or SplitId protocol calls only during their corresponding frame. To generate micropayments, a payer will wait for the beginning of the next epoch – only then will it contact the bank with the Withdraw, OpenAC and SplitId call, made at random within the corresponding frames of the current epoch. A payer cannot engage in a InitChain or Spend protocol during this epoch, but instead it will wait for the next epoch. This approach requires the payers to be roughly time synchronized with the bank, yet it can be easily made tolerant to clock skews by time gaps between Withdraw, OpenAC and SplitId frames.

Improvement: Probabilistic identity share distribution. To further reduce the communication and computation cost of the SplitId and Spend protocols, instead of providing one share for each payment, a probabilistic mechanism can be deployed in which payers provide an identity share with a certain probability p for each payment. For example, if each payment is one cent and the payer's account contains \$10, then for $p = 1\%$ a new share will be offered by the payer (on average) to the payee for each spent dollar (100 cents) – we say the share is "worth" \$1. This lottery-like payment scheme can significantly reduce communication overheads in the SplitId protocol. While payees will now need to keep track of the number of identity shares received per payment amount (e.g., above at least 1 share per dollar on average), the protocol is statistically fair. Moreover, payers can always protect their privacy by stopping before giving out too many shares.

C. Properties

The security properties of PlusPay are defined in the following, similar to the ones introduced in Section IV-B for the previous protocols. Since PlusPay relies on two additional procedures, OpenAC and SplitId, its security properties are slightly different from their counterparts defined in Section IV-B.

Payment Unlinkability:

- $\mathcal{A}$ generates and sends the public key of the bank pk_B, T_e, the length of bank epochs, m, the standard currency amount in payments and n, the total number of shares to $\mathcal{C}$. $\mathcal{C}$ runs UKGen for two users U_0 and U_1 and sends their public keys to $\mathcal{A}$.
- $\mathcal{C}$ generates two anonymous accounts AC_0 and AC_1, each with a balance m, and two corresponding payments P_0 and P_1 by calling Withdraw, OpenAC and SplitId for U_0 and U_1. All the calls are made in a single epoch. The calls for U_0 and U_1 for each operation are done in random order. For instance, $\mathcal{C}$ selects a bit b_o and calls OpenAC for U_{b_o} first.
- $\mathcal{C}$ selects a bit $b \in_R \{0, 1\}$ and calls InitChain for P_b with $\mathcal{A}$ from account AC_b. $\mathcal{C}$ then executes Spend m times with $\mathcal{A}$ for payment P_b.
- A outputs its guess b' for b.

The advantage of $\mathcal{A}$ is defined as Pr[$b' = b$] $-1/2$.

Payment Indistinguishability:

- $\mathcal{A}$ generates and sends pk_B, T_e, m and n to $\mathcal{C}$. $\mathcal{C}$ runs UKGen for two users U_0 and U_1 and sends their public keys to $\mathcal{A}$.
- $\mathcal{C}$ generates two anonymous accounts AC_0 and AC'_0 and two payments P_0 and P'_0, by running Withdraw, OpenAC and SplitId on behalf of user U_0, with the adversary $\mathcal{A}$. $\mathcal{C}$ similarly generates two anonymous accounts AC_1 and AC'_1 and two payments P_1 and P'_1 on behalf of U_1. All the calls are made in a single epoch - in random order for U_0 and U_1. Note that at the end of this step payment each anonymous account holds one of the payments generated.
- $\mathcal{C}$ selects two bits $b_1, b_2 \in_R \{0, 1\}$ and runs InitChain and Spend up to m times with $\mathcal{A}$ for both payments P_{b_1} and P'_{b_2} - using accounts AC_{b_1} and AC'_{b_2}.
- $\mathcal{A}$ outputs its guess b for $b_1 \oplus b_2$.

The advantage of $\mathcal{A}$ in this game is defined as Adv($\mathcal{A}$) = Pr[$b = b_1 \oplus b_2$] $-1/2$.

Balance: The adversary $\mathcal{A}$ runs Withdraw, followed by OpenAC and SplitId l times with the bank B. Let S_j be the set of identity shares generated during the jth run $1 \leq j \leq l$. Let $S = \{S_j | 1 \leq j \leq l\}$. Let P_j be the payment generated in the jth run. $\mathcal{A}$ then outputs a deposit tuple D such that IdShare $\notin S$. The advantage of $\mathcal{A}$ is defined to be $Adv(\mathcal{A}) = Pr[Deposit(\mathcal{A}(params_{\mathcal{A}}, D), B(params_B)) = 1]$.

Exculpability:

- **Setup**: $\mathcal{A}$ generates and sends the bank's public key, pk_B, its own public key, $pk_{\mathcal{A}}$ and T_e, the length of bank epochs, m, the standard currency amount in payments and n, the total number of shares to $\mathcal{C}$. $\mathcal{C}$ runs UKGen for a user U and sends its public key to $\mathcal{A}$. The following step is then executed l times.
- **Query**: $\mathcal{C}$ interacts with $\mathcal{A}$ by calling Withdraw, OpenAC and SplitId on behalf of user U – it generates an anonymous account AC_i and a payment P_i of m coins under the serial number of AC_i. $\mathcal{C}$ interacts with $\mathcal{A}$ by calling InitChain once and Spend up to m times on P_i.
- **Success Criterion**: $\mathcal{A}$ outputs a serial number SN and a proof Prf.

$\mathcal{A}$ succeeds if Verify (U, SN, Prf) accepts.

D. Analysis

Correctness: If an honest user runs Withdraw with an honest bank, the bank's verifications will succeed since the payment messages are correctly generated. Similarly, OpenAC will succeed, since the e-cash is valid. SplitId also succeeds since the token TK is correctly signed and was never used before and the identity shares were correctly generated. If an honest payer runs InitChain with an honest payee, the payee's verifications will succeed: P and CMT are valid. Similarly, the payee's verifications for Spend will succeed since the identity shares, their commitments and the indexes of the revealed shares are correctly generated. Finally, if an honest user runs Deposit with an honest bank, the bank will accept the micro-coins: if the user's verifications during InitChain and Spend succeeded, so will the bank's verifications executed during Deposit.

Payment Unlinkability: The PlusPay protocol provides micropayment unlinkability by adding one level of indirection. Payments are made from and can be linked to anonymous accounts, which are unlinkable to their owners. Let us assume that there exists a PPT algorithm $\mathcal{B}$ that has advantage ϵ_{PU} in the payment unlinkability game. Following a Withdraw call, a user obtains values EC and TK. An adversary that can link EC and TK to a user with non-negligible advantage has a non-negligible advantage against the blindness property of the blind signature protocol. Following an OpenAC call, the user obtains a bank signed balance certificate, BalCert and an anonymous account AC. An adversary that can link AC and BalCert to a user with non-negligible advantage, has the same advantage in linking input to output messages in a mix network (see Section III-A). Similarly, an adversary cannot have non-negligible advantage in linking the payment P to a user following a SplitId call, without having the same advantage against the blindness property of the blind signature solution or against the hiding property of the commitment scheme. The InitChain and Spend protocols are called using only the keys associated with the anonymous account AC_b. None of the data available to $\mathcal{B}$ during either InitChain or Spend (i.e. bank signed commitments, balance certificate, hash chain root commitment) can be linked to the payer's identity but rather only to AC_b. Then, if $\mathcal{B}$ has advantage ϵ_{PU} in the payment unlinkability game, we can construct a PPT algorithm $\mathcal{A}$ that can break the Hiding property of TSS – using the same reduction employed in the proof of Theorem 1. The PlusPay solution allows payers to balance an efficiency-privacy trade-off. Specifically, a payer may choose to re-use an anonymous account for multiple payment chains, at the expense of micropayment indistinguishability. This may be often desirable and more efficient when payment indistinguishability is not of concern. If, on the other hand, a payer chooses to not re-use anonymous accounts, then PlusPay also satisfies the payment unlinkability game.

Payment Indistinguishability: Let us assume that there exists a PPT algorithm $\mathcal{B}$ that has advantage ϵ_{PI} in the payment indistinguishability game. $\mathcal{B}$'s advantage can be from linking an anonymous account (AC_{b_1} and AC_{b_2}) to a user (U_0 or U_1) or from determining whether AC_{b_1} or AC_{b_2} belong to the same user. The first case contradicts the payment unlinkability property (see above). The second case can be if values EC, AC, BalCert or P from different executions of Withdraw, OpenAC and SplitId can be distinguished as being from the same user with non-negligible advantage. If this is the case, then an adversary also has advantage ϵ_{PI} against the blindness property of the blind signature protocol, or against the the hiding property of the commitment scheme or in linking input to output messages in a mix network. The InitChain and Spend protocols are called using only the keys associated with the corresponding anonymous accounts. If the user does not overspend, $\mathcal{B}$ can be used to build an adversary that has advantage ϵ_{PI} in the hiding game of a TSS – using a similar reduction to the one employed in the proof of Theorem 2.

In addition, PlusPay provides payee anonymity.

Payee Anonymity: Let us assume that an algorithm $\mathcal{B}$ exists that has advantage ϵ_{PA} in the Payee Anonymity game (see Section IV-B). The challenger $\mathcal{C}$ interacts with $\mathcal{B}$ during InitChain and Spend and obtains a deposit tuple D. $\mathcal{C}$ uses D as an input to the Deposit procedure it runs with $\mathcal{B}$ over AChan. At the end of Deposit, $\mathcal{C}$ obtains a signed payment token of value $l - f + 1$. $\mathcal{B}$ cannot have a non-negligible advantage in guessing whether $\mathcal{C}$ has acted as user V_1 or user V_2, without being able to build input-to-output correspondence in a mix net or without breaking the blindness property of the blind signature protocol.

Balance: Consider an adversary $\mathcal{A}$ that has a non-negligible advantage in the Balance game defined in Section VII-C. $\mathcal{A}$ is then able to generate a deposit tuple D such that IdShare $\notin S$ and the Deposit procedure succeeds with non-negligible probability. Then, $\mathcal{A}$ is able to either (i) forge a value $P = \{C\}_{sk_B}$, where C is the set of identity share commitments or (ii) produce a value IdShare $\notin S$ such that $CMT(IdShare) \in C$. Case (i) cannot occur without $\mathcal{A}$ having a non-negligible advantage in forging B's signature. Case (ii) can only occur if $\mathcal{A}$ has non-negligible advantage in the partially blind signature protocol or if $\mathcal{A}$ has non-negligible advantage in defeating the binding property of the commitments scheme.

Culpability: Let us assume that an adversary generates a micropayment chain of m coins and runs Spend more than m times. Then, Theorem 3, that also holds for PlusPay, shows that the adversary chance of spending m or less shares is at most $f \ln \frac{f}{f-1} - 1$, where $f = n/m$, n being the total number of identity shares – a system parameter. If B has more than m shares, it can reconstruct the identity of the over spender.

Exculpability: Let us assume that an algorithm $\mathcal{B}$ for PlusPay exists that has non-negligible probability of succeeding in the exculpability game defined in Section IV-B. Then, we can build an adversary $\mathcal{A}$ that that has non-negligible advantage in the hiding game of a TSS. The reduction is similar to the one used for CoinPay. $\mathcal{A}$ interacts with the challenger $\mathcal{C}$ in the hiding game, to generate values R_0 and R_1 and to obtain two sets of shares of either R_0 or R_1. $\mathcal{A}$ uses each set of shares in the Query step of the exculpability game with $\mathcal{B}$, to generate and spend coins. The coins are withdrawn from user accounts with ids R_0 or R_1. However, the OpenAC and SplitId generate anonymous accounts for both users. If a Query step fails (see above proofs), $\mathcal{A}$ repeats it (a polynomial number of times in t and m). Let ϵ_S be the advantage of $\mathcal{B}$, $\epsilon_S = Adv(\mathcal{B}, 2)$, of producing a serial number SN and a proof P such that Verify(R_b,SN,P) accepts, for $b \in \{0, 1\}$. Then, with probability ϵ_S, $\mathcal{A}$ returns 0 to $\mathcal{C}$: its guess is that the two sets of shares were for the same number (R_0 or R_1).

Other Properties: By construction, the bank can be offline during transactions: payees can verify the validity of received micropayments without querying the bank. Since PlusPay extends a classic micropayment solution, it is straightforward to see that coins from the same payment chain can be aggregated. Finally, Section VIII presents an evaluation of PlusPay's overheads and shows that they are low.

VIII. Performance Evaluation

We have evaluated the performance of the CoinPay and PlusPay on off-the shelf end-user hardware: Intel P4, 3.4 GHz, 2GB RAM, openssl 0.9.8b [43]. Under light-load multi-user mode, this setup allows about 261 RSA-1024 signatures and 5423 RSA verifications per second as well as more than 1.5 million SHA-1 crypto hashes per second (on 16Byte blocks). We assumed a network of no more than 6Mbps bandwidth and 1ms latency. Typical Tor latencies were assumed (500ms) [2]. We estimated overheads and throughputs for the case of a payer opening an anonymous account and depositing 100 coins (while generating one identity share per coin).

We have replaced the commitment set C employed by both solutions with a more efficient Merkle tree built on the

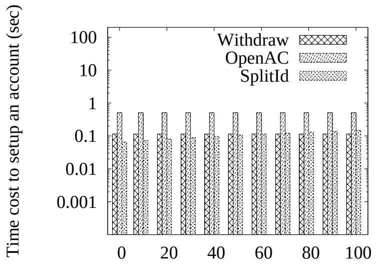

Fig. 1. PlusPay Account setup as a function of the overspending control factor (f). OpenAC has the highest cost, dominated by the Tor latency.

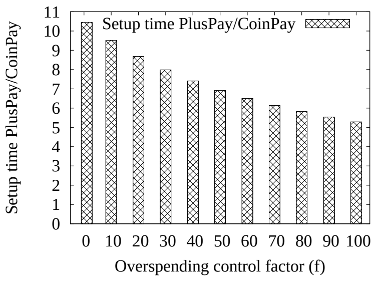

Fig. 2. The ratio of the setup costs of PlusPay and CoinPay decreases with increasing f. PlusPay needs only a few anonymous account re-uses to become more efficient than CoinPay.

commitments of identity shares from C. This enables the bank to sign a single value (the root of the Merkle tree). The proof of correctness of a commitment consists of revealing the Merkle tree path corresponding to that commitment.

Payment Setup. Figure 1 shows the costs for each payment setup protocol call, when the overspending control factor (f) increases from 1 to 100. The y-axis is shown in logarithmic scale. For the cut and choose step of the Withdraw protocol, we have considered that the payer generates $t = 100$ messages ($2t - 1$ RSA blinding operations), out of which the bank signs only one (one RSA signature). Even though the network delay of Withdraw depends on $t + 1$ messages and one challenge/response protocol, the total overhead of the protocol is around 100ms.

During the OpenAC step, the computation overhead consists of the bank performing an e-cash verification (one RSA verification) and a RSA signature generation. The total cost is then dominated by Tor (around 500ms). The cost of generating a new key pair is not factored in as it can be incurred offline by the client. The SplitId protocol consists of the payer generating $t = 100$ identity sets and building a Merkle tree over each set (2 $f * m$ crypto hashes, where m is the payment value). This is followed by a cut and choose protocol consisting of $2t - 1$ RSA encryptions, $t - 1$ share reconstructions and one RSA signature. The reconstruction can be done efficiently using an $O(m \log^3 m)$ algorithm [44], [45]. The network delay of SplitId is dominated by the cost of sending the t blinded identity sets. Figure 1 shows that, as expected, the overall cost of SplitId increases linearly with the overspending control factor f. This increase is reasonable, ranging from less than 100ms for $f = 1$ to no more than 200ms for $f = 100$.

In CoinPay, the setup consists of a single call of the SignMC protocol. Using the previous evaluation scenario, Figure 2 shows the ratio between the setup time of PlusPay and the

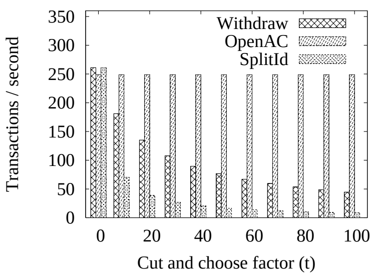

Fig. 3. The number of operations the bank can process in a second when t (in the cut and choose protocols) ranges from 1 to 100. Even for $t = 100$, the bank can perform 10 SplitId/SignMC calls/s.

setup time of CoinPay. For small values of the overspending control factor, the ratio is around 10 (CoinPay is faster here). The ratio decreases for higher values of f, reaching 5 for $f = 100$. This decrease is due to the fact that the SignMC protocol is very similar and has the same cost as the SplitId protocol of the PlusPay solution. Both protocols generate identity shares whose number is determined by f. Thus, higher values of f make the setup stages of PlusPay and CoinPay converge. This ratio also shows the number of times an anonymous account generated in PlusPay has to be reused before the cost of its generation becomes smaller than the cost of using CoinPay. This was one of the main advantages of PlusPay over CoinPay. Our evaluation shows that this number is small, effectively minimizing usage pattern leaks (see Section VII-D).

Throughputs. Figure 3 shows the computation overhead for the bank during Withdraw, OpenAC and SplitId protocol calls of PlusPay when the number of message duplicates, t, during the cut and choose protocols increases from 1 to 100 but the value of f is set to 10. The cost of the SignMC protocol of CoinPay is the same as the cost of SplitId of PlusPay. The OpenAC protocol has constant overhead, allowing the bank to process around 250 OpenAC calls per second. The overhead of the Withdraw and SplitId/SignMC protocols is linear in the value of t. That is, the bank can process between 50 (for $t = 100$) and 260 (for $t = 1$) Withdraw calls per second. SplitId/SignMC are more compute intensive – the bank can perform between 10 (for $t = 100$) and 260 (for $t = 1$) calls per second. As a result then, for PlusPay the length of the SplitId time frame has to be around 5 times the length of the Withdraw time frame. Note that the OpenAC time frame can be as small as a fifth of the Withdraw time frame.

Costs. InitChain consists of a signature generation and m crypto hashes performed by the payer and three signature verifications, performed by the payee. Spend consists roughly of $\log{(f * m)}$ crypto hashes performed by the payee. Deposit requires the bank to perform one signature verification and $\log{(f * m)}$ crypto hashes per micropayment to verify the correctness of the identity shares.

Figure 4 shows the transaction cost (InitChain plus Spend) and the Deposit cost per micropayment. While for short chains, the transaction cost is higher (5ms for 1 payment chain) than the deposit cost (2ms), this changes for longer chains. The cost of a Deposit operation is dominated by a signature verification, whereas for a micropayment transaction, signature verification

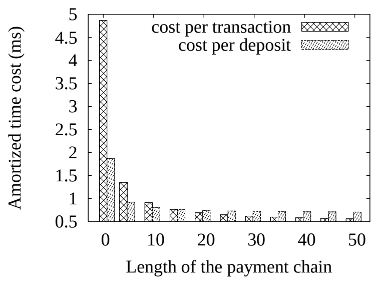

Fig. 4. Cost of micropayment transactions and deposit operations for PlusPay and CoinPay, as a function of the micropayment chain length: 500 μs for a transaction and 750μs for a Deposit (chain of length 50).

costs are amortized over the number of spent micropayments. For a chain of length 50, the transaction cost is close to 500μs and the deposit cost is 750μs – even for short chains the transaction cost is almost negligible.

IX. Conclusions

We introduced the first set of efficient and correct micropayment mechanisms with anonymity. They feature offline verification, aggregation, statistical overspending prevention and very low overheads. Throughputs of thousands of transactions per second are supported. We implemented ORPay. In our experiments, its overheads are under 4%. Anonymous micropayments become thus a viable incentive mechanism for practical deployment in networked services such as packet routing, anonymizers, and peer to peer file sharing, enabling fairness, quality of service and global cost optimization.

References

[1] Apple. Apple iTunes Music Store. Online at http://www.apple.com/itunes, 2009.
[2] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.
[3] Beverly Yang and Hector Garcia-Molina. Ppay: micropayments for peer-to-peer systems. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 300–310, New York, NY, USA, 2003. ACM.
[4] Yao Chen, Radu Sion, and Bogdan Carbunar. Xpay: practical anonymous payments for tor routing and other networked services. In *WPES '09: Proceedings of the 8th ACM workshop on Privacy in the electronic society*, pages 41–50, New York, NY, USA, 2009. ACM.
[5] Ronald L. Rivest and Adi Shamir. Payword and micromint: Two simple micropayment schemes. In *Proceedings of the International Workshop on Security Protocols*, pages 69–87, London, UK, 1997. Springer-Verlag.
[6] Charanjit Jutla and Moti Yung. Paytree: amortized-signature for flexible micropayments. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, Oakland CA, 1996.
[7] Ronald L. Rivest and Silvio Micali. Electronic lottery tickets as micropayments. In Rafael Hirschfeld, editor, *Financial Cryptography*, pages 307–314, Anguilla, British West Indies, 1997. Springer.
[8] Mark S. Manasse. The millicent protocols for electronic commerce. In *WOEC'95: Proceedings of the 1st conference on USENIX Workshop on Electronic Commerce*, pages 9–9, Berkeley, CA, USA, 1995. USENIX Association.
[9] Charalampos Manifavas Ross J. Anderson and Chris Sutherland. NetCard: A practical electronic-cash system. *Lecture Notes in Computer Science - Security Protocols*, 1189:49–57, 1997.
[10] Phillip M. Hallam-Baker. World Wide Web Consortium: Micro Payment Transfer Protocol (MPTP). Online at http://www.w3.org/TR/WD-mptp-951122, 1995.
[11] Richard J. Lipton and Rafail Ostrovsky. Micro-payments via efficient coin-flipping. In *In Financial Cryptography*, pages 1–15. Springer-Verlag, 1998.
[12] Ralf Hauser, Michael Steiner, and Michael Waidner. Micro-payments based on iKP. Technical report, 1996.
[13] Elli Androulaki, Mariana Raykova, Shreyas Srivatsan, Angelos Stavrou, and Steven M. Bellovin. Par: Payment for anonymous routing. In Nikita Borisov and Ian Goldberg, editors, *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, pages 219–236, Leuven, Belgium, July 2008. Springer.
[14] Amir Herzberg. Micropayments. In *Payment technologies for E-commerce*, pages 245–282, 2003.
[15] I. Papaefstathiou and C. Manifavas. Evaluation of micropayment transaction costs. *Journal of Electronic Commerce Research*, 5(2):99–113, 2004.
[16] David Chaum. Blind signatures for untraceable payments. In *Proceedings of CRYPTO '82*, pages 199–203, 1982.
[17] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
[18] David Chaum. Privacy protected payments: Unconditional payer and/or payee untraceability. In *Proceedings of SmartCard 2000*, 1988.
[19] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *CRYPTO '88*, pages 319–327, London, UK, 1990. Springer-Verlag.
[20] David Chaum and Torben Pryds Pedersen. Transferred cash grows in size. In *EUROCRYPT'92: Proceedings of the 11th annual international conference on Theory and application of cryptographic techniques*, pages 390–407, Berlin, Heidelberg, 1993. Springer-Verlag.
[21] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In *CRYPTO '93: Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, pages 302–318, 1994.
[22] Matthew K. Franklin and Moti Yung. Secure and efficient off-line digital money (extended abstract). In *ICALP '93*, pages 265–276, London, UK, 1993. Springer-Verlag.
[23] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. In *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'95, pages 209–219, Berlin, Heidelberg, 1995. Springer-Verlag.
[24] George I. Davida, Yair Frankel, Yiannis Tsiounis, and Moti Yung. Anonymity control in e-cash systems. In *FC '97*, pages 1–16, London, UK, 1997. Springer-Verlag.
[25] Jan Camenisch, Ueli M. Maurer, and Markus Stadler. Digital payment systems with passive anonymity-revoking trustees. In *Proceedings of ESORICS '96*, pages 33–43, London, UK, 1996. Springer-Verlag.
[26] Yair Frankel, Yiannis Tsiounis, and Moti Yung. "indirect discourse proof": Achieving efficient fair off-line e-cash. In *ASIACRYPT '96*, pages 286–300, London, UK, 1996. Springer-Verlag.
[27] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Proceedings of EUROCRYPT*, pages 302–321, 2005.
[28] Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. Endorsed e-cash. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 101–115, 2007.
[29] Johnny Ngan, Roger Dingledine, and Dan Wallach. Building incentives into Tor. Technical report, 2008.
[30] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.
[31] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2), 1981.
[32] Masayuki Abe. Universally verifiable mix-net with verification work indendent of the number of mix-servers. In *Proceedings of EUROCRYPT*, pages 437–447, 1998.
[33] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 248–259, 1994.
[34] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In *ASIACRYPT*, pages 252–265, 1996.
[35] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.
[36] David Chaum. Blind signatures system. *Advances in Cryptology, Proceedings of CRYPTO*, pages 153–156, 1983.
[37] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
[38] George R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, 1979.
[39] Holger Bürk and Andreas Pfitzmann. Digital payment systems enabling security and unobservability, 1989.
[40] Kai Wei, Alan J. Smith, Yih-Farn Robin Chen, and Binh Vo. Whopay: A scalable and anonymous payment system for peer-to-peer environments. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 13, Washington, DC, USA, 2006. IEEE Computer Society.
[41] Larry Shi, Bogdan Carbunar, and Radu Sion. Conditional e-cash. In *Proceedings of Financial Cryptography*, pages 15–28, 2007.
[42] Marina. Blanton. Improved conditional e-payments. In *Applied Cryptography and Network Security*, pages 188–206. Springer, 2008.
[43] OpenSSL. The openSSL project. OpenSSL: The open source toolkit for SSL/TLS. www.openssl.org.
[44] Donald E. Knuth. *Fundamental Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, second edition, 10 January 1973.
[45] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.