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

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. However, assuming the existence of such an always available bank is unreasonable in many distributed scenarios. Yet, while sacrificing some accuracy for efficiency may seem reasonable, the cost of privacy is inestimable.

Bogdan Carbunar is with the School of Computing and Information Sciences at Florida International University, Miami, FL. E-mail: carbunar@gmail.com. Yao Chen and Radu Sion are with the Stony Brook Computer Science Department, Stony Brook, NY. E-mail: {yaochen,sion}@cs.stonybrook.edu.

Here we introduce 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.

II. Related Work

In [4], we introduced two micropayment mechanisms – ORPay and PlusPay. In this paper, we extend the work with a new protocol, CoinPay. 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. We further add formal definitions and proofs for the three solutions.

Numerous micropayment schemes exist, including 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] and PAR [13]. Due to space limitation, in the following we detail the ones most related to our schemes.

We base our constructions on PayWord [5]. PayWord deploys hash chains to model payment sessions and requires only one signature per session. 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.

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
KEY PROPERTIES OF MICROPAYMENT SOLUTIONS.

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.

Table I summarizes the key properties of several existing micropayment schemes. For a detailed discussion on related work please see [14], [4].

III. TOOLS

We require several cryptographic primitives: (i) a semantically secure [15] encryption scheme, (ii) an unforgeable signature scheme and (ii) a random oracle G. We further use the following tools.

Anonymizers: Mix networks [16], [17], [18], [2]. 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) they operate correctly, i.e., outputs correspond to a permutation of the inputs and (ii) they provide privacy, i.e., an observer is not able to determine which input element corresponds to a given output element better than guessing.

Blind Signatures: Blind signatures allow a user to obtain a signature from a signer, where (i) the signer does not learn information about the signed message – *blindness* and (ii) the user cannot obtain more than l signatures after l runs of the signing protocol – *unforgeability*.

Threshold Secret Sharing (TSS): A (k, n) TSS schemes [19], [20] ensures *hiding*: An adversary (provided with access to a TSS oracle) controlling the choice of two values R_0 and R_1 and given less than k shares of R_b ($b \in_R \{0, 1\}$) can guess the value of b with probability only negligible higher than 1/2.

Commitment Schemes: A commitment scheme is a triple $(Gen, CMT, Open)$. Gen generates a public commitment key, CMT produces a commitment value for m and Open takes as input a commitment value and additional information and produces either a message or outputs error. A commitment scheme is correct if Open(CMT(m)) = m. A commitment scheme needs also provide *hiding* and *binding* properties. Informally, hiding implies that it is hard for any PPT adversary $\mathcal{A}$ to generate two messages such that $\mathcal{A}$ can distinguish between their commitments. Binding implies that it is hard for any PPT adversary $\mathcal{A}$ to find two messages whose commitments are equal (collision).

IV. MODEL

Operation. Let B denote the "bank", any authority that manages payment accounts. 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. 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.

A. Anonymous Micropayments

We define an *anonymous micropayment scheme* to be a set of protocols μP = { BKGen, UKGen, Withdraw, InitChain, Spend, Deposit, Verify }.

- **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 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 U and 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 U and V after the InitChain protocol succeed 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 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 V and B and allows V to deposit a set $D = (CMT, w, k)$ containing k micro-coins into its bank account, using the last coin w and the commitment CMT. If w is not valid or CMT 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 $Id(U)$, Verify accepts the proof. Otherwise, it outputs ERROR.

Initially, each participant runs UKGen (or BKGen by the bank) to generate its private and public parameters. Users call Withdraw to obtain micropayments from their bank account. To spend m coins with a vendor V, user U initiates the InitChain procedure followed by m runs of the Spend procedure. A user deposits micropayments in her bank account

by running the Deposit procedure. Deposit is also used by payers to redeem unspent micro-coins. Verify can be run by any participant to verify overspending claims.

B. Properties

We now define a set of security properties for anonymous micropayment mechanisms, inspired from [21].

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.

We define the notion of payee anonymity, describing the inability of an adversary to guess the identity of a user performing a deposit operation [22], [23], [24], [25]. 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.
- $\mathcal{C}$ (as V_b) runs Deposit for the m coins received from $\mathcal{A}$.
- $\mathcal{A}$ 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: 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 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 serial no. SN and proof P.

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 B 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.

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 anonymity 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 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);
```

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 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[1] 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.

[1]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.

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 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 [2].

VI. CoinPay: Overspending Protection

CoinPay aims to provide protection against overspending. Payers explicitly withdraw cash from their accounts before being able to generate micropayments. Additionally, instead of directly signing micropayment chains, and thus revealing their identity, payers ask the bank to partially blindly sign the w_0 roots of the micropayment chains. This To prevent a payer from overspending, threshold splitting is employed to generate shares of the payer's identity. These "identity shares" are 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. In the case of overspending, the bank will have enough shares to reconstruct the identify the over spender.

A. The CoinPay Protocols

Withdraw$(U(pk_B, sk_U, m), B(sk_B, m))$**.** U runs Chaum's partially blind signature protocol [26] 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 $n > m$ [3] 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 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.

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.

[2]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.

[3]There is a relationship between n and m that will be defined later, e.g., in the proof of Theorem 2.

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.

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: $\epsilon_{TSS} = \epsilon_{CP}$. ■

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 2: 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. We now introduce PlusPay, a protocol addressing these problems.

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. 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 CMT, the BalCert(AC) certificate and the P value to V. V does the following:

- 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 μCHN.
- 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. Else it adds $IdShare_i$ to set S. Let the deposit tuple $D = (P, CMT, BalCert, S, R_U, w_f, w_l, f, l)$, where w_f is the first and w_l the 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}. Also, their commitments 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. A payer can call Deposit to redeem unspent micropayments and her identity will be protected as long as she 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.

A detailed analysis of PlusPay's properties is included in [27].

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 [28]. 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 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 (a) 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, its total overhead is only 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 [29], [30]. The network delay of SplitId is dominated by the cost of sending the t blinded identity sets. Figure 1 (a) 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 1 (b) shows the ratio between the setup time of PlusPay and the 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 **??**).

Throughputs. Figure 1 (c) 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 1 (c) 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

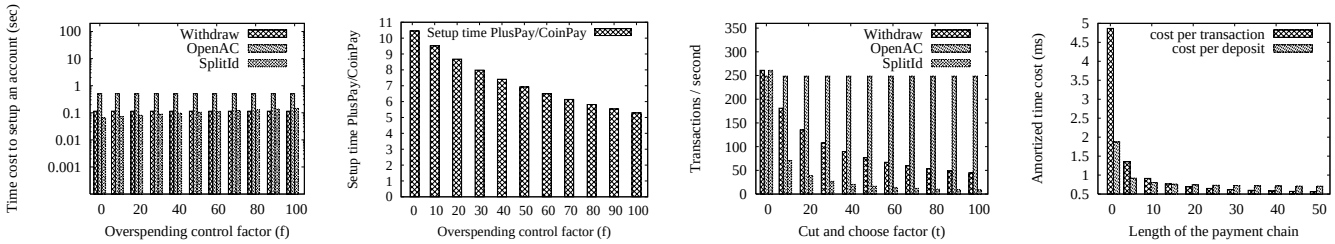

Fig. 1. (a) PlusPay Account setup as a function of the overspending control factor (f). OpenAC has the highest cost, dominated by the Tor latency. (b) The ratio of the setup costs of PlusPay and CoinPay decreases with increasing f. PlusPay needs only a few anonymous account re-uses to become more efficient than CoinPay. (c) 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. (d) 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).

signature verification, whereas for a micropayment transaction, signature verification 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] Yao Chen, Radu Sion, and Bogdan Carbunar. Tipping pennies? privately. practical anonymous micropayments. Online at http://www.cs.stonybrook.edu/~sion/research.

[15] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[16] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2), 1981.

[17] Masayuki Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In *Proceedings of EUROCRYPT*, pages 437–447, 1998.

[18] 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.

[19] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[20] George R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, 1979.

[21] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Proceedings of EUROCRYPT*, pages 302–321, 2005.

[22] Holger Bürk and Andreas Pfitzmann. Digital payment systems enabling security and unobservability, 1989.

[23] 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.

[24] Larry Shi, Bogdan Carbunar, and Radu Sion. Conditional e-cash. In *Proceedings of Financial Cryptography*, pages 15–28, 2007.

[25] Marina. Blanton. Improved conditional e-payments. In *Applied Cryptography and Network Security*, pages 188–206. Springer, 2008.

[26] David Chaum. Blind signatures system. *Advances in Cryptology, Proceedings of CRYPTO*, pages 153–156, 1983.

[27] Radu Sion Bogdan Carbunar, Yao Chen. Tipping Pennies? Privately. Practical Anonymous Micropayments. Technical report, Florida International University, 2012. http://www.cs.fiu.edu/~carbunar/upayments_techrep.pdf.

[28] OpenSSL. The openSSL project. OpenSSL: The open source toolkit for SSL/TLS. www.openssl.org.

[29] Donald E. Knuth. *Fundamental Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, second edition, 10 January 1973.

[30] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

Bogdan Carbunar is an assistant professor in the School of Computing and Information Sciences at the Florida International University. Previously, he held various researcher positions within the Applied Research Center at Motorola. His research interests include distributed systems, security and applied cryptography. He holds a Ph.D. in Computer Science from Purdue University.

Yao Chen is a Ph.D student at Stony Brook University under the supervision of Radu Sion. Her research interests include Efficient Computing and Cyber Security. She received her B.S. from Zhejiang University.

Radu Sion heads the Stony Brook Network Security and Applied Cryptography (NSAC) Lab. His research interests include Information Assurance and Efficient Computing. He builds systems mainly, but enjoys elegance and foundations, especially if of the very rare practical variety. Sponsors and collaborators include IBM, IBM Research, NOKIA, Xerox, as well as the National Science Foundation which awarded also the CAREER Award. Radu is on the steering board and organizing committees of conferences such as NDSS, Oakland S&P, CCS, USENIX Security, SIGMOD, ICDE, FC and others.