# XPay: Practical anonymous payments for Tor routing and other networked services

Yao Chen
Stony Brook Network Security and Applied Cryptography Lab
Stony Brook, NY 11794
yaochen@cs.sunysb.edu

Radu Sion*
Stony Brook Network Security and Applied Cryptography Lab
Stony Brook, NY 11794
sion@cs.sunysb.edu

Bogdan Carbunar
Motorola Labs
1295 E. Algonquin Rd. IL05
Schaumburg, IL 60195
carbunar@motorola.com

## 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 ORPay for Tor), provide full anonymity (both for payers and payees), and support thousands of transactions per second.

## Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General

## General Terms

Security, Performance

## Keywords

Anonymity, Privacy, Micropayments, Onion routing

## 1. 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 [16] and P2P file sharing [33] 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 protocols. 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 this would be inefficient and probably not anonymous etc. 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. 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 quickly after no more than a tunable small amount of cash has been involved. In such a case only, perpetrator identities can be revealed. The mechanisms are practical with minimal overheads and support thousands of transactions per second.

## 2. PRELIMINARIES

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

*The authors are supported in part by the NSF through awards CNS-0627554, CNS-0716608, CNS-0708025 and IIS-0803197. The authors also wish to thank Xerox/Parc, Motorola Labs, IBM Research, the IBM Software Cryptography Group, CEWIT, and the Stony Brook Office of the Vice President for Research.


*WPES'09,* November 9, 2009, Chicago, Illinois, USA.


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

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 $k$. $X \hookrightarrow_R D$ is a random choice of value $X$ from domain $D$. The notation $Pr_X(Y)$ denotes the probability of event $Y$ given the input $X$.

## 2.1 Tools

We require several cryptographic primitives with all the associated semantic security [19] properties: (i) a secure, collision resistant hash function which builds a distribution from its input that is indistinguishable from a uniform random distribution (we use the notation $H(x)$), (ii) a semantically secure [19] encryption (a computationally bounded adversary has a negligible advantage at determining whether a pair of encrypted items of the same length represent the same or unique items), (iii) *Merkle (hash) trees* [26], and (iv) a pseudo random number generator (PRNG) whose output is indistinguishable from a uniform random distribution over the output space (we use the notation $G(x)$). We note that assumptions (i) and (iv) are equivalent in both semantics and execution costs.

We assume the existence and security of cut and choose blind signature protocols [10][2]. We assume the existence of a simple threshold splitting mechanism [19, 24, 20] that takes as input a value $X$ and generates $n$ "shares" thereof, such that any $m+1$ or more shares are enough to re-construct $X$. No probabilistic polynomial time algorithm exists that can re-construct $X$ out of less than $m+1$ shares.

## 2.2 Payment Chains.

A *micropayment scheme* is a set of protocols $\mu P = \{BKGen, UKGen, InitChain, Spend, Deposit\}$.

*Payment chains* were first introduced in PayWord [30] by Rivest and Shamir. A payer generates a long one-way hash chain, where the end-element ("the root") 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 [30] are as follows.

**BKGen**$(1^k, params)$ The bank $B$ invokes this to generate its public/private key pair, $(pk_B, sk_B)$.

**UKGen**$(1^k, params)$ Each payer $U$ calls UKGen to generate its public/private key pair, $(pk_U, sk_U)$.

**InitChain**$(U(sk_U), V(sk_V, pk_B))$. This is run between $U$ and $V$ to allow $U$ to initialize a micropayment chain. The protocol is implemented as follows. Let $w_m \hookrightarrow_R \{0,1\}^k$ be a random number chosen by $U$. $U$ generates a micropayment chain $\mu CHN = w_0, w_1 ... w_m$ as discussed above, 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. $U$ commits to the chain by sending $V$ the value $CMT = \{V, w_0, D, \ldots\}_{sk_U}$, where $D$ is an expiration date, and a bank signed certificate. $V$ verifies the certificate and $U$'s signature on the $CMT$ value. If this fails, $V$ generates ERROR.

**Spend**$(U(sk_U, \mu CHN, k, n), V(sk_V, CMT, k, n))$. This is run between a payer $U$ and a payee $V$ after the InitChain protocol has completed and $k$ micro-coins from $CHN$ have been spent by $U$ to $V$. It allows $U$ to spend another $n$ micro-coins to $V$. To this end $U$ sends to $V$ the $n$th PayWord, $w_n$. $V$ verifies that $h^n(w_{k+n}) = w_k$. If it fails, $V$ generates ERROR.

**Deposit**$(V(sk_V, CMT, w_k, pk_B), B(pk_V, sk_B))$. This is run between a payee $V$ and the bank $B$ and allows $V$ to deposit $k$ micro-coins into its bank account. $V$ sends the commitment $CMT$ and 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 $k$.

## 2.3 Adversaries and Deployment.

**Adversary.** We assume a computationally bounded PPT adversary that may collude with or masquerade as any number of vendors, payers and the bank. Banks are trusted to perform the bank-side protocol correctly (even though they might be curious). This adversary is more powerful than the one assumed in Tor – to preserve anonymity, Tor assumes the adversary does not control the first and the last node in a circuit. Moreover Tor only guarantees unlinkability of parties and not full anonymity (we define this as "transaction $k$-unlinkability" below). It is also at least as strong as in related work [4] where "users can only observe the traffic going through them and a limited amount of the rest of the network traffic".

**Deployment.** Naturally, micropayments should be *efficient*: computation and communication costs should be much lower than transacted values. Execution times should allow acceptable high transaction rates. Bank involvement should be minimal. For most practical applications this means that the Bank should not be required to be online at transaction time. Moreover, it might be desirable to allow multiple micropayments to be aggregated into a single large payment which can be cashed in one operation.

*Correctness* mandates preventing illicit spending and coin forgery. Often, for efficiency, these requirements can be relaxed to only upper-cap the amount of illicitly handled cash before probabilistic detection thereof. This is appropriate especially given the "micro" nature of the payments' values.

Finally, to be useful, often micropayment schemes need to provide *anonymity*, a property that is quintessential for more traditional e-cash but seems harder to achieve here due to efficiency requirements. Payers' privacy is not the only concern here. Payments should also be un-traceable to payees and different payments from the same payer should not be linkable. This is **a stronger definition of anonymity for payments** than in existing research [7, 6] – most importantly because it also considers payee's privacy.

[2]Naturally, we are aware of the existence of more efficient blind signature schemes but we need the cut and choose properties. We note that cut and choose has been used for digital payment mechanisms before [10].

**Properties.** The above can be summarized as follows.

**P1.1. Payment $k$-unlinkability.** Given a micropayment instance $p$ and the set $\mathcal{U}$ of $k$ active users, there is no PPT algorithm $Alg$ that can distinguish the payer from $\mathcal{U}$ that has generated the micropayment. That is, the following value is negligible [19] $|Pr_{u \hookleftarrow_R \mathcal{U}}(Alg(u, p, [B])) - 1/k|$.

**P1.2. Transaction $k$-unlinkability.** Given a micropayment instance $p$, the payee $V$ that has been paid with $p$, and the set $\mathcal{U}$ of $k$ active users, there is no PPT algorithm $Alg$ that can distinguish the payer from $\mathcal{U}$ that has spent $p$ with $V$. That is, the following value is negligible $|Pr_{u \hookleftarrow_R \mathcal{U}}(Alg(u, p, [B], [V])) - 1/k|$, even when $Alg$ has access to the knowledge of the bank and the payee.

**P1.3. Payment $k$-indistinguishability.** Given a micropayment chain instance $p$, the set $\mu\mathcal{P}$ of existing $k$ valid (non-double spent) micropayment chains, the set $\mathcal{U}$ of all users, and a polynomial function $f$, there is no PPT algorithm $Alg$ that can distinguish $p$ from other payments in $\mu P$, some of which are potentially generated by the same payer. That is, the following value $|Pr_{p' \hookleftarrow_R \mu\mathcal{P}}(Alg(p, p', [B], [\mathcal{U}])) - 1/f(k)|$ is negligible,even when $Alg$ has access to the Bank's and all the users' knowledge.

**P1.4. Payee $k$-unlinkability.** Given a micropayment instance $p$ and the set $\mathcal{U}$ of $k$ active users, there is no PPT algorithm $Alg$ that can distinguish the payee from $\mathcal{U}$ that deposits $p$ with the bank. That is, the following value is negligible $|Pr_{u \hookleftarrow_R \mathcal{U}}(Alg(u, p, [B])) - 1/k|$, even when $Alg$ has access to the knowledge of the bank.

**P2. Offline verification.** The solution should not require the online availability of any participant, including any trusted party such as the bank. This implies that the payee should be able to verify the validity of the payment without having to interact with the bank.

**P3. 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.

**P4. Double and Over Spending Prevention** No payment instance should be spent more than once and no participant should be able to spend more than its account balance (either through *coin forgery* or otherwise). If double (spending same payment multiple times) or overspending (spending more coins from the payment than allowed) occurs, the identity of the culprit should be revealed. For efficiency, in this paper we are re-formulating this in terms of a *tunable statistical upper-bound on the amount that can be double/over-spent before detection*.

**P5. Low overheads.** The micropayment transaction protocols have to be computation and communication efficient relatively to their deployment environment.

We will describe the XPay protocols below in the context of PayWord, introduced in Section 2.2. PayWord is a good starting point because it naturally satisfies **P2**-**P5**. During Spend transactions, the bank is offline (**P2**). Since the spender's identity is included in the payment, overspending is natively discouraged and immediately discovered (**P4**). Moreover, **P3** is satisfied since payees only redeem the last micropayment received. Finally, the costs include one signature generation and verification per micropayment chain and one hash computation per micropayment spent. Thus, **P5** is also satisfied.

Yet, PayWord does not satisfy anonymity properties **P1.1** - **P1.4**. 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.

# 3. BACKGROUND

**Micropayments.** A number of micropayment schemes have been proposed. These include PayWord [30], MicroMint [30], PayTree [21], Peppercorn [29], Millicent [25], Netcard [3], Lipton and Ostrovsky's coin flipping-based scheme [23], Payfair [34], PPay [33], PAR [4]. In the following we will detail a few.

PayWord [30] deploys hash chains for implementing the basic unit payment and only require a signature per session as detailed in section 2.2. Payments can be aggregated. To prevent overspending, the payer identity is included in the payments, thus defeating anonymity. PayTree [21] is building upon PayWord to further reduce the number of required expensive crypto signatures by building a Merkle Tree to efficiently authenticate multiple chains.

MicroMint [30] 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 [29] "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 [33] – 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.

**E-cash.** The use of blind signatures and of the cut-and-choose protocol for providing untraceable electronic cash payments was proposed in [9] [11] [12] [13]. The problem of transferable e-cash was analytically studied first by Chaum and Pedersen [14]. The work of Brands [5] 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 [18] 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 [32] [15] [8] [17]. Stadler et al. [32] introduced the notion of coin tracing and introduced several tracing mechanisms, requiring the trustee to be on-line at withdrawal. Camenisch et al. [8], Frankel et al. [17] and Davida et al. [15] 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.

Camenish et al. [6] 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 [7] Camenish 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 [16] 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, compact and endorsed e-cash are likely not suitable for micro-transactions. For instance, the endorsed e-coin generation algorithm requires multiple modular exponentiations, being thus several orders of magnitude costlier than the XPay suite (cryptographic hash evaluations). This is essential for micropayments.

Simon [31] proposes a simple e-cash protocol in a network where anonymous communication is possible. The payer generates the e-cash by having the bank sign $f(x)$ where $x$ is the payer's secret and $f$ is a one-way function. The e-cash can be transferred by revealing $x$ to the payee. The payee can then either cash the money with the bank or further transfer it by providing the bank with $x$ and asking it to sign $f(y)$ for which it knows $y$. If the communication between the payee and the bank is anonymous, the payee remains anonymous and can transfer the money further. The bank can link the start and end points of a transfer chain, however, for long chains this information may be meaningless. Moreover, the end point of a transfer chain may repeat this protocol with itself, to artificially increase the length of the chain.

A recent result, PAR [4] 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. PAR 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). Finally, expensive modular arithmetic operations for each individual payment coin are unnecessary and often impractical in scenarios requiring high throughputs – a very probable deployment for micropayments.

Ngan et al. [27] 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. And personal traffic initiated from those 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 place a big burden on the directory servers.

While a number of the above micropayment mechanisms satisfy some of the correctness and efficiency properties outlined in Section 2.3, in the following we introduce a first set of practical micropayment mechanisms with increasing degrees of anonymity.

# 4. ORPAY: ONION ROUTING PAYMENTS

We start by investigating the use of micropayments in Tor [16] 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 through a "banking" service provided by Tor's directory. The accounts' balance can 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 routers whose balance exceeds a threshold.

We first note that Tor only guarantees unlinkability of parties and not full anonymity (we named this transaction $k$-unlinkability **P1.2**). 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.

## 4.1 Protocol

The protocol proceeds as follows. Initially, the bank service runs $BKGen$ to establish system parameters and each participant, either Tor client or router calls $UKGen$ to generate its key pair.

Next, a user who needs to send data through Tor (the source) starts by creating a circuit consisting of $n$ routers ($n$ defaults to 3). Tor builds circuits incrementally; in a first step the source creates a connection to the first Tor router. This is the stage where the two (the source and first router) run $InitChain$ to establish a micropayment chain for future use. The first router then extends the circuit to the second router and similarly these two routers run $InitChain$ to establish another micropayment chain. This process is performed $n$ 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).

During the actual data transfer, *conceptually* the source will include $n$ 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 $n$ micropayments are part of the chain initialized during the circuit establishment step. Without loss of generality we also assume that forwarding a *Relay* packet[3] is worth one micropayment. The first router then piggybacks $n-1$ micropayments, from its own micropayment chain, to each packet forwarded to the second router. This process is continued until the last router receives one micropayment from the preceding router, to forward the packet to the destination.

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. This can be done by having the source piggybacking micropayments to ACK packets traveling back to the destination. 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, ACK 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 $W * e * n$, where $e$ is the value of each payment amount and $n$ 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 (that are not routers) are not given rank and they have the lowest priority in the Tor network.

## 4.2 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 and 0.5ms for 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. To this end 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% as expected 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.

# 5. PLUSPAY: FULL ANONYMITY

As we discussed, by its very nature, the above reputation/incentive mechanism will not hide payers or payees identities. By letting each router pay to its successor, and considering the fact that routers can be simultaneously part of multiple circuits, it provides the transaction $k$- unlinkability **P1.2**. This is however no longer true for other internet services such as those offered by cloud computing or even P2P file sharing, where there are no intermediate nodes to hide the sender's identity.

In this section we introduce a solution (we call it PlusPay) that provides full anonymity **P1.1- P1.4**. The PlusPay protocol does not require the existence of intermediate nodes and allow payers to make untraceable payments to payees. It 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 anonymity property **P1.4**. 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. While not considered by previous e-cash or micropayment technologies,

[3]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.

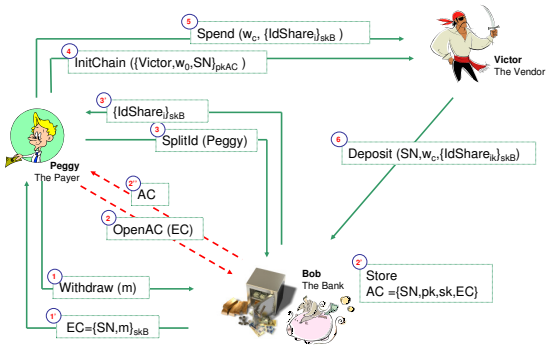


**Figure 1: Overview of PlusPay. Victor is a payee. Green (full) lines arrows denote communication performed over authenticated channels. Red (dotted) arrows denote anonymized communications.**

this property can be essential from the perspective of hidden (anonymous) services [16].

Note that anonymous accounts may enable illegal activities such as money laundering. While it is outside the scope of this paper, we note that additional mechanisms may be used to allow "trusted" third parties to reveal identities of anonymous account holders in special cases. The anonymous account in PlusPay is mainly intended to prevent privacy leaks in daily transactions, both for individual users and service providers.

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. Once the number of revealed shares exceeds $m$ (in the case of overspending), the bank will be able to identify him.

## 5.1 Overview

Overall, this 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 $w_0$, the payer will sign it using the private key associated with its anonymous account. Here micropayment chains are similar to checks drawn from bank-hosted anonymous account (Figure 1).

## 5.2 Solution

More formally, PlusPay, a micropayment system is a set of protocols, PlusPay = {BKGen, UKGen, Withdraw, OpenAC, SplitId, InitChain, Spend, Deposit}.The BKGen and UKGen protocols are inherited from PayWord. We now focus on the new logic.

**Withdraw**$(U(pk_B, sk_U, m), B(pk_U, sk_B, m))$**.** This is run between a payer $U$ and the bank $B$ and enables the payer to withdraw an e-cash bill of value $m$ from its account, and a bank signed one-time use token, in an unlinkable fashion. The e-cash will be used to open an anonymous account in **OpenAC** and the token will be used later as a proof of an account with balance $m$ in **SplitId**.

The protocol works as follows. $U$ engages in a cut and choose blind signature protocol with $B$. Specifically, the payer generates $t$ bills and $t$ tokens of the format $(SN_i, m)$ and $token_i, m$, for $i = 1..t$, where $SN_i$ and $token_{id}$ are independent random numbers (using different formats to be distinguished). The bank then signs a blinded version of a randomly chosen $(SN_i, m)$ bill, and $token_i, m$ token, while verifying the "well-formed"-ness of the remaining $t-1$ of them. The verification consists of checking that $SN$s and $token$s are in the right formats and the value of the second field of the revealed messages is $m$. If it fails, $B$ generates ERROR. Otherwise, the payer 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 $B$.

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 e-cash bills of other values. 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.

**OpenAC**$(U(pk_B, sk_U, EC, m), B(sk_B, m))$**.** This protocol is run between a payer and the bank through an anonymizer $AChan$ and allows the payer to deposit e-cash of value $m$ into a newly generated anonymous account, unlinkable to the payer. The output for the payer is either an anonymous account $AC$ with a balance $m$ or ERROR. The bank output is a transaction record or ERROR.

This is implemented as follows. $U$ first generates a public/private key pair $(pk_{AC}, sk_{AC})$ for the account $AC$. Then $U$ contacts $B$ through $AChan$ and presents the blindly bank-signed e-cash $EC$ obtained during the previous (**Withdraw**) step and $pk_{AC}$. $B$ checks the validity of the e-cash (the signature and whether it has been spent before). If EC is invalid, $B$ generates ERROR. Otherwise, $B$ opens an anonymous account $AC$ identified by a (random) serial number $SN_{AC}$ and seeds it with the $m$-valued currency of $EC$. It then signs $BalCert(AC) = \{pk_{AC}, SN_{AC}, m\}_{sk_B}$ a balance certificate, and sends back the anonymous account information $AC = \{SN_{AC}, m, BalCert\}$ to $U$ through $AChan$.

We note that this is the only step requiring an anonymizer. 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 apparent circularity of new payers being unable to pay for anonymizer traffic when joining.

**SplitId**$(U(pk_B, sk_U, AC, TK, m), B(sk_B, m))$**.** The SplitId protocol allows the payer to obtain the bank's blind signature on a set of shares of its identity. Its output for $U$ is a Threshold Identity Revealing Anonymous Account (TIRAC, see below) or ERROR. The output for $B$ is a record of the

transaction or ERROR. $U$ uses a $(m, n)$ threshold splitting mechanism to split $Id(U)$ into $n > m$ shares, such that any but no less than $m + 1$ shares can be used to reconstruct $Id(U)$. The format of the identity shares is

$$IdShare_i = \{sh_i, i, m, \{h(SN_{AC} \oplus R_U)\}_{sk_{AC}}\}$$

where $R_U$ is a random value. First, $U$ needs to present $TK$ to $B$ to prove that an account with $m$ balance exists. $B$ verifies the signature on $TK$ and $TK$ has not been used before. If all checks verify, $B$ records the *token* to prevent reuse. Then the payer then builds a Merkle tree on the set of shares (let $r_0$ be the tree's root) and engages in a blind signature protocol with $B$ to (i) retrieve $B$'s blind signature on the root of the Merkle tree of the shares and (ii) allow $B$ to verify the "well-formed"-ness of the shares. The bank's output, if an ERROR is not encountered, is its signature on the last blinded (unrevealed during the cut-and-choose protocol) Merkle tree root – let it be denoted $r_0$. $U$'s output is then $TIRAC = \{SN_{AC}, \{r_0\}_{sk_B}\}$.

The purpose of the $\{h(SN_{AC} \oplus R_U)\}_{sk_{AC}}$ value is to link the identity shares to a valid anonymous account, used later in the Spend protocol. Since this step is performed here over an authenticated channel, the bank is not allowed to see the anonymous account's serial number (this is ensured by the encryption of $SN_{AC}$). During the Spend protocol however, $U$ will be able to prove the identity share's link to the anonymous account, by revealing to $SN_{AC}$ and $R_U$ to the payee.

Note that the payee has to verify $\{h(SN_{AC} \oplus R_U)\}_{sk_{AC}}$ just once per chain. For any subsequent identity shares, no public key cryptography is needed – a simple binary string match to the previously authenticated $\{h(SN_{AC} \oplus R_U)\}_{sk_{AC}}$ value is sufficient.

**InitChain**$(U(sk_U, BalCert(AC), m, \{r_0\}_{sk_B}), V(sk_V, pk_B, m))$**.** Similar to PayWord, this allows a payer $U$ to initialize a payment session with a payee $V$. For this, $U$ 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 $AC$, $CMT = \{V, w_0, SN_{AC}\}_{sk_{AC}}$. $U$ then sends the commitment $CMT$, the $BalCert(AC)$ certificate and the root of the Merkle tree, $\{r_0\}_{sk_B}$, to $V$. $V$ validates the commitment by checking that (i) the public key $pk_{AC}$ contained in $BalCert$ can verify the signature used on the $CMT$ value and (ii) the account number $SN_{AC}$ contained in $CMT$ is consistent with the one in $BalCert$. $V$ also verifies $B$'s signature on the identity share Merkle tree root. If any of these checks fails, $V$ returns ERROR. Otherwise, **Spend** can be launched.

**Spend**$(U(sk_U, pk_V, TIRAC, \mu CHN), V(sk_V, pk_B))$**.** This implements the same functionality as the PayWord Spend protocol. It is run between $U$ and $V$ after InitChain has completed and $k$ micro-coins have been spent by $U$ to $V$. It allows $U$ to spend an additional $n$ micro-coins with $V$. While sending $V$ a new micropayment the payer $U$ also sends a provably randomly selected identity share together with the (Merkle) proof that the share is from the original set of shares authenticated by the tree root signed by $B$. The payer then reveals the $SN_{AC}$ and $R_U$ values to allow $V$ to verify the identity share's link to the anonymous account. Let the chosen share be $IdShare_i = \{sh_i, i, m, \{h(SN_{AC} \oplus R_U)\}_{sk_{AC}}\}$. Next $V$ checks the "well-formed"-ness of the share, specifically that (i) the share and its proof correctly reconstruct the $r_0$ value, (ii) it has the expected index $i = G(Id(V), w_k, \ldots)\ mod\ n$ (this can be done only once per chain, see note above), (iii) the balance $m$ is consistent in IdShare and BalCert and (iv) the hash of the xor of $SN_{AC}$ and $R_U$ is indeed signed with the secret key corresponding to the public key included in the BalCert certificate. If any of these checks fails, $V$ generates ERROR. Otherwise, it accepts payment.

**Deposit**$(V(sk_V, CMT, w_k, pk_B), B(pk_V, sk_B))$**.** In addition to proving to $B$ knowledge of the commitment value $CMT$ and of $k$ micro-coins, the payee needs also present $k$ different $IdShare$ values. $B$ verifies that all the identity shares are associated with the same anonymous account $AC$. If the check fails, $B$ generates ERROR. Otherwise, it records the shares associated with the serial number $SN_{AC}$. 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.

The payee can choose to stay anonymous by requesting $B$ to deposit the micro-coins in its anonymous account and by initiating this protocol over an anonymous channel. Similarly, a payer can redeem unspent micropayments using its anonymous account and pretending to be a vendor.

If $B$ detects any overspending with an anonymous account $AC$, it will be able to reconstruct the owner's identity using the identity shares recorded.

## 5.3 Improvements

**Improvements: 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 goes through 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.

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.

While this solves the timing attack issue probabilistically, now apparently an actively malicious adversarial bank can now also try to target specific victims e.g., by only answering their Withdraw calls and filtering out all other calls. This would effectively allow it to trace the Withdraw, OpenAC and SplitId calls made by the victims. While our threat model does not include such an actively malicious bank.

## 5.4 Analysis

By construction, the bank can be offline during transactions (**P2**). Payees can verify the validity of received micropayments without querying the bank. Moreover, payments from the same payment chain can be aggregated (**P3**). We evaluate overheads in Section 5.5 and show they are low. Overspending can be controlled (**P4**):

THEOREM 1. *The expectation of overspending is controllable by the bank.*

*Proof (sketch):* For an anonymous account with balance $m$, $m+1$ different identity shares ($m+1$ payments) are enough to reconstruct the identity of the payer. Let there be a total of $n = f * m$ identity shares, where $f > 1$ is a system wide parameter called the *overspending control factor*. Note that every time a payer spends, payees randomly request a share from the pool of $n$. Then let $x$ be the expected number of times a payer can spend from an anonymous account, until reveal $m+1$ different identity shares. This reduces to the classical Coupon Collector's Problem.

$$\begin{aligned} E[x] &= E[1] + E[2] + \cdots + E[m+1] \\ &= \frac{n}{n} + \frac{n}{n-1} + \cdots + \frac{n}{n-m} \\ &\approx f \ln \frac{f}{f-1} * m \end{aligned}$$

For instance for $f = 10$, $E(x) \approx 1.05m$, meaning that overspending can be controlled below 5%.

□

PlusPay also satisfies **P1.1 -P1.4**. Let $N$ be the total number of system users, $T_w$ the expected per-payer interval between withdrawals, and $T_e$ the length of the bank generated time epochs. Then

THEOREM 2. *Micropayments generated using PlusPay are $k$-unlinkable, with $k = N * T_e/2T_w$.*

This basically shows that a payer $U$ cannot be linked to one of its micropayments or anonymous accounts during Withdraw, OpenAC or SplitId.[4] We are now proving that even when the payee is an adversary, the InitChain, Spend and Deposit protocols can neither be used to provide such a link, nor to link $U$ to the payee.

THEOREM 3. *The micropayment transactions of an honest payer of PlusPay are $k$-unlinkable, where $k > N*T_e/2T_w$, even with payee-bank collusion.*

The PlusPay solution allows payers to balance an efficiency privacy trade-off. Specifically, a payer may chose to re-use an anonymous account for multiple payment chains, at the expense of micropayment $k$-indistinguishability (**P1.3**). This may be often desirable and more efficient when **P1.3** is not of concern. If, on the other hand, a payer chooses to not re-use anonymous accounts, then **P1.3** holds:

THEOREM 4. *Micropayments generated in the PlusPay solution by non - overspending payers are $k$ - indistinguishable, where $k$ is the minimum between the total number of anonymous accounts with available funds and $(N * T_e/2T_w)$.*

[4]Due to space limitations, we can not accommodate the proofs of Theorem 2 - 5. However they will be included in the future journal version of the paper.

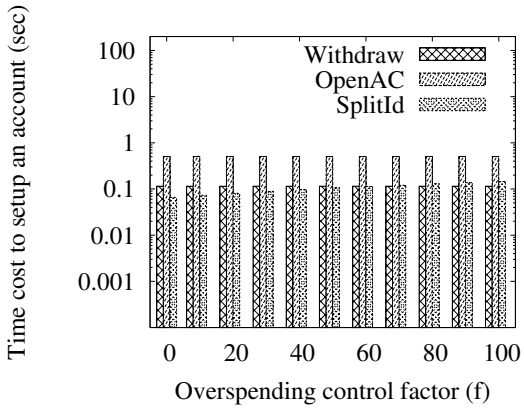


**Figure 2: PlusPay setup step. (a) PlusPay Account setup: consisting of Withdraw, OpenAC and SplitId procedure calls, as a function of the overspending control factor ($f$). The highest cost is due to OpenAC – dominated by the Tor latency. The SplitId's dependence on the $f$ value is linear, but reasonable (less than 200ms for $f = 100$).**

Moreover, PlusPay also provides payee anonymity. That is, even when the payer and the bank collaborate, they are unable to link deposited payments to the payee.

THEOREM 5. *Payees depositing micropayments with the bank in PlusPay are N - indistinguishable, where N is the number of active users in the system (with an active anonymous account).*

## 5.5 Performance Evaluation

We have evaluated the performance of 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) [16]. For illustration purposes, 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).

**Payment Setup.** For the PlusPay solution, payment setup consists of three protocol calls, Withdraw, OpenAC and SplitId. Figure 2(a) shows the costs for each 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 consists of sending $t + 1$ messages and one challenge/response protocol, the total overhead of the protocol is around 100ms.

During the OpenAC step, the dominating network latency is due to Tor. 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 gener-

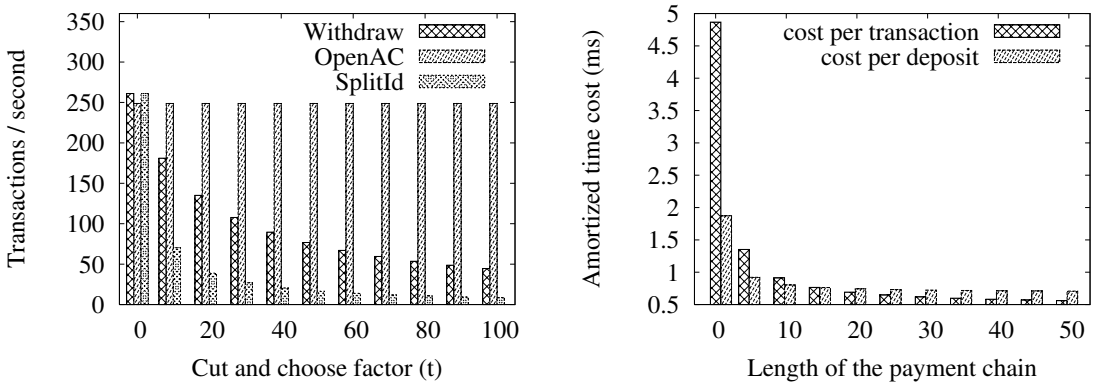


**Figure 3: PlusPay throughputs and performance. (a) The number of Withdraw, OpenAC and SplitId/SignMC calls that the bank can process in a second when the number of message duplicates, $t$, in the cut and choose protocols ranges from 1 to 100. SplitId/SignMC are the protocols with the highest bank overhead, however, even for $t = 10$, the bank can perform 10 SplitId/SignMC calls per second. (b) The cost of micropayment transactions and deposit operations for PlusPay, as a function of the micropayment chain length. Even for short chains these costs are almost negligible ($500\mu$ s for a transaction and $750\mu$s for a Deposit for a chain of length 50).**

ating 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 [22, 2]. The network delay of SplitId is dominated by the cost of sending the $t$ blinded identity sets. Figure 2(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 $f = 100$.

**Throughputs.** Figure 3(a) 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 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.** During a micropayment transaction between a payer and a payee, InitChain and Spend are invoked. Later, the payee calls Deposit to redeem the micropayments. 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 3(b) 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 costs are amortized over the number of spent micropayments. Note that for a chain of length 50, the transaction cost is close to $500\mu$s and the deposit cost is $750\mu$s. Thus, even for reasonably short chains the transaction cost is almost negligible.

# 6. 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 of thousands of transactions per second are possible. Especially we implemented ORPay and the experiments show it only added less than 4% overhead. 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.

# 7. REFERENCES

[1] Apple iTunes Music Store. Online at `http://www.apple.com/itunes`.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[3] R. Anderson, C. Manifavas, and C. Sutherland. NetCard: A practical electronic-cash system. *Lecture Notes in Computer Science - Security Protocols*, 1189:49–57, 1997.

[4] E. Androulaki, M. Raykova, S. Srivatsan, A. Stavrou, and S. M. Bellovin. Par: Payment for anonymous routing. In N. Borisov and I. Goldberg, editors,

*Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, pages 219–236, Leuven, Belgium, July 2008. Springer.

[5] S. Brands. Untraceable off-line cash in wallets with observers (extended abstract). In *CRYPTO*, 1993.

[6] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *Ronald Cramer, editor, Advances in Cryptology – Eurocrypt*, volume 3494, 2005.

[7] J. Camenisch, A. Lysyanskaya, and M. Meyerovich. Endorsed e-cash. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, 2007.

[8] J. Camenisch, U. M. Maurer, and M. Stadler. Digital payment systems with passive anonymity-revoking trustees. In *ESORICS '96: Proceedings of the 4th European Symposium on Research in Computer Security*, pages 33–43, London, UK, 1996. Springer-Verlag.

[9] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology— Proceedings of Crypto '82*, pages 199–203. Plenum Press, 1982.

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

[11] D. Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[12] D. Chaum. Privacy protected payments: Unconditional payer and/or payee untraceability. In *Proceedings of SmartCard 2000*, 1988.

[13] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO '88: Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology*, pages 319–327, London, UK, 1990. Springer-Verlag.

[14] D. Chaum and T. P. Pedersen. Transferred cash grows in size. In *EUROCRYPT*, 1992.

[15] G. I. Davida, Y. Frankel, Y. Tsiounis, and M. Yung. Anonymity control in e-cash systems. In *FC '97: Proceedings of the First International Conference on Financial Cryptography*, pages 1–16, London, UK, 1997. Springer-Verlag.

[16] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[17] Y. Frankel, Y. Tsiounis, and M. Yung. "indirect discourse proof": Achieving efficient fair off-line e-cash. In *ASIACRYPT '96: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pages 286–300, London, UK, 1996. Springer-Verlag.

[18] M. K. Franklin and M. Yung. Secure and efficient off-line digital money (extended abstract). In *ICALP '93: Proceedings of the 20th International Colloquium on Automata, Languages and Programming*, pages 265–276, London, UK, 1993. Springer-Verlag.

[19] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[20] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO '95: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, pages 339–352, London, UK, 1995. Springer-Verlag.

[21] C. Jutla and M. Yung. Paytree: amortized-signature for flexible micropayments. In *Second USENIX Workshop on Electronic Commerce*, Oakland CA, 1996.

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

[23] R. J. Lipton and R. Ostrovsky. Micro-payments via efficient coin-flipping. In *In Financial Cryptography*, pages 1–15. Springer-Verlag, 1998.

[24] A. Lysyanskaya and C. Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 331–350, London, UK, 2001. Springer-Verlag.

[25] M. 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.

[26] R. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Research in Security and Privacy*, 1980.

[27] J. Ngan, R. Dingledine, and D. Wallach. Building incentives into Tor. Technical report, 2008.

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

[29] R. L. Rivest. Electronic lottery tickets as micropayments. In R. Hirschfeld, editor, *Financial Cryptography*, pages 307–314, Anguilla, British West Indies, 1997. Springer.

[30] R. L. Rivest and A. 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.

[31] D. R. Simon. Anonymous communication and anonymous cash. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 61–73, London, UK, 1996. Springer-Verlag.

[32] M. Stadler, J.-M. Piveteau, and J. Camenisch. Fair blind signatures. In *Proceedings of EUROCRYPT*, pages 209–219, 1995.

[33] B. Yang and H. 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.

[34] S. Yen, J. Lee, and J. Lee. Payfair: A prepaid internet micropayment scheme promising customer fairness. In *Proc. of International Workshop on Cryptographic Techniques and E-Commerce, CrypTEC 99*, pages 213–221, 1999.