Offline Micropayments without Trusted Hardware

Matt Blaze[1], John Ioannidis[1], and Angelos D. Keromytis[2]

[1] AT&T Labs - Research
180 Park Avenue
Florham Park, NJ 07932 USA
{mab,ji}@research.att.com

[2] Computer Science Department, Columbia University
1214 Amsterdam Avenue, M.C. 0401
New York, NY 10027 USA
angelos@cs.columbia.edu

Abstract. We introduce a new micropayment scheme, suitable for certain kinds of transactions, that requires neither online transactions nor trusted hardware for either the payer or payee. Each payer is periodically issued certified credentials that encode the type of transactions and circumstances under which payment can be guaranteed. A risk management strategy, taking into account the payers' history, and other factors, can be used to generate these credentials in a way that limits the aggregated risk of uncollectable or fraudulent transactions to an acceptable level. These credentials can also permit or restrict types of purchases. We show a practical architecture for such a system that uses a Trust Management System to encode the credentials and policies. We describe a prototype implementation of the system in which vending machine purchases are made using consumer PDAs.

Keywords: Trust Management; Risk Management; Microbilling; Payments; Digital Cash.

1 Introduction

Current electronic payment systems are not well matched to occasional, low-valued transactions. (For the purposes of this discussion, we use the term "electronic payment system" broadly, to encompass conventional credit cards, stored-value cards, online and offline digital cash, *etc.*)

A central requirement for any electronic payment system is that a single compromise or failure should not have catastrophic consequences. For example, it should not be possible to double spend in a digital cash system, nor should the compromise of a client's authorization secret entail unlimited client liability or uncollectible transactions. Traditional payment systems are designed to *prevent* such failures. Unfortunately, the prevention mechanisms are generally too expensive to support occasional, low-valued transactions. Typically, such systems require online transactions, trusted client hardware such as smartcards, or must assume conditions that are not always true, such as that payers can be held responsible for any and all fraud or misuse of their authorization secrets.

In this paper, however, we present a new approach that focuses instead on *risk management*. Our central observation is that in some applications we can relax many of the expensive requirements associated with electronic payment systems while still keeping fraud or uncollectible transactions within acceptable levels. We shift the security functions performed by online authorization of transactions to certified code that can authorize offline transactions under certain conditions. These conditions are customized to each client according to a risk management strategy customized to the application.

There are three main contributions in this paper. First, we describe a framework in which certified offline authorizations created by a risk management strategy replace online authorizations for occasional, low-valued transactions. We then describe an architecture for a practical payment system in which a trust management system [BFL96] is used to encode the client risk management strategy. Finally, we describe a prototype implementation based on the KeyNote trust management toolkit [BFIK99], in which users can purchase vending machine items using credentials stored on conventional palmtop computers.

1.1 Related Work

Most currently-used protocols for Internet e-commerce are based on credit card charging over SSL [Hic95]. Such schemes require the merchant to perform a "hidden" (from the user's point of view) online credit check. The cost of such checks can be on the order of 10 (US) cents, making them expensive for low-value transactions. The more recently developed SET [SET] and CyberCash protocols [EBCY96] do not address this issue.

NetBill [CTS95] is a transactional payment protocol with many advanced features (atomicity, group membership, pseudonyms, *etc.*) that requires communication with the NetBill server for each transaction, thus exhibiting the same drawback with respect to micropayments as the simpler online protocols already mentioned. Other general-purpose payment protocols [NM95, BGH$^+$95, FB98] are unattractive for micropayments for these same reasons.

Digital cash-based systems [Cha82, Cha92, MN94, BGJY98, dST98] provide many desirable features (potentially total anonymity, inherent off-line operation), but do not directly address the issue of double-spending (fraud). Some e-cash systems use online checking (thus negating the off-line operation capability). Others rely on detection after the fact, which introduces the potential for large-scale simultaneous multiple-spending. The same drawback is manifest in several micropayment procotols, such as PayWord [RS], PayTree [JY96], micro-iKP [HSW96], and others [Tan95]. While the double-spending possibility is an inherent property of all such systems, none of the above protocols employ any kind of risk management scheme to address it.

NetCents [PHS98] and Millicent [Man95] are scrip-based off-line-friendly micropayment protocols. As the monetary unit used in these protocols is vendor-specific, double-spending is made very difficult (if not impossible). The assumption behind both protocols is that people tend to re-use the same merchants repeatedly. If this assumption holds, the interactions between the customer and the bank are kept at a minimum. A hidden assumption is that merchants have "total information" over their sales, so double-spending with the same merchant is detectable. If the merchant has many distinct points of sale, the potential for double-spending is re-introduced, unless continuous communication and database synchronization is maintained between the different points. This would consequently negate the benefits of off-line operation.

IBM's MiniPay [HY96, Her98] uses a protocol that is somewhat similar to that described in this paper. MiniPay was developed primarily for use within a web browser, and a lot of effort has gone into the user interface aspect. Risk management is implemented as a decision to perform an online check with the billing server based on the total spending by the customer that day, and some parameter set by the merchant. The billing provider cannot customize risk-management parameters on a per-customer and/or per-merchant basis.

Person-to-person (P2P) payment systems, such as PayPal or X.com (now merged), allow users to exchange money online. Typically, the provider's web server needs to be contacted and an instruction issued for a money transfer. In that respect, the transaction is very similar to a bank wire transfer. There also exist modules that allow users to directly exchange money through palmtop computers. Such systems typically have no built-in security mechanisms; in the best of circumstances, they are a straight variant of offline digital cash.

While our system can operate on its own, it could also be integrated in some type of electronic wallet, such as SWAPEROO [DBGM$^+$98].

Finally, the use of a PDA as an electronic wallet is not new. [DB99] describes an implementation of the PayWord system for the PalmPilot.

1.2 Offline Transactions and Risk Management

Consider a simplified view of how transactions are processed in traditional credit- or debit- card systems. Each payer has an account with a card issuer, against which charges can be made up to some limit. When a user wants to charge a transaction, she provides her account number to the merchant, who calls the card issuer for authorization. The card issuer checks that the transaction is less than the available balance, and if so, subtracts the transaction amount from the balance and authorizes the transaction. The user also signs an authorization to charge her account for the transaction amount. At periodic intervals, each merchant sends the signed authorizations they have collected to the card issuer, which transfers appropriate funds to the back to the merchants and bills the cardholders' accounts accordingly. (Real credit and check card authorization and clearing mechanisms are more complicated than this in practice, but still follow approximately this basic procedure).

Observe that even though settlement and clearing can be (and are) done offline and in batches, each transaction still requires an online authorization. This step is needed for two reasons, both related to the general-purpose design of the credit / check card model. First, losses from uncollectible transactions are limited by imposing an account limit on each user, which must be checked and debited with each transaction to prevent credit overruns or negative balances. Secondly, the account number must be checked to be sure that it is actually associated with an account in good standing, to prevent fraud from stolen or forged account numbers. (Observe that the online transaction could, in principle, be eliminated if the card holder can be given trusted hardware, such as a smartcard, that maintains its own state about the status of the account and produces signed transaction authorizations).

Such systems become extremely vulnerable to fraud and abuse if the online transaction authorization (or the trusted hardware) is eliminated. (For example, consider why few retailers will accept checks from random customers without verifying their validity with the bank). There would be no limit to what the account holder could spend during the validity period of the card, and no mechanism to detect invalid or forged account numbers. Clearly, such a vulnerability is unacceptable in a general-purpose payment system.

We observe that in some applications, however, it is acceptable to risk the occasional uncollectible or fraudulent transaction if, in the aggregate, tolerating the losses costs less than preventing them through online transactions or deploying secure hardware. In fact, assuming risk and tolerating loss is the basis upon which credit systems work.

The basic idea behind our scheme is that we include with the user's account identifier certified information that describes the circumstances under which transactions can be authorized offline. These circumstances would differ from application to application, and indeed, from user to user, but are selected in a way that makes it difficult to profitably exploit or abuse a compromised account. The rules are designed to allow offline authorization for those transactions where fraud is unlikely and in which the cost of an online authorization is greater than the value of the transaction itself.

We assume that each user's authorization data is managed by a small, portable device of modest computational ability and with some capacity for communication, such as infrared or low-power radio. Ideally, the device is something the user already owns for some other purpose; PDAs and cellular telephones are especially good examples.

In this scheme, the users' credentials manage and limit risk in several ways. First, credentials would have a limited lifetime, perhaps a day or two, and would have to be refreshed by communicating with the issuer at regular intervals. The validity period would be determined by the length of time the issuer is willing to tolerate loss from stolen client devices, and also by the natural interval that the user is likely to be able to communicate back with the issuer.

Because each transaction is authorized by trust management credentials, the system can also be used in applications where the ability to conduct certain kinds of transactions must be restricted in various ways and where different treatment is given to different classes of customers. For example, some transactions (alcohol, tobacco, pornography, binding contracts, *etc.*) might be restricted to adults; it is a simple matter to encode a requirement for an "adulthood" credential in the vendor's policy. Similarly, certain transactions might require licenses or special permission (dangerous goods, car rental, medical supplies), which are also easy to encode as credentials and check for in a policy. Conversely, the credential-based mechanism also makes it easy to create restricted forms of money that can only be used for certain things, such as social welfare food stamps or spending money given by parents to their children.

The most important mechanism for limiting fraud and abuse is the transaction limit encoded in the credentials. The kinds of transactions permitted are determined according to the risk management strategy of the account issuer, and are designed to limit the usefulness to a thief of a compromised user's credentials and secrets. For example, an encoded strategy might permit the offline purchase of newspapers, but only a few copies from any given vendor. If a user's device is stolen, the thief would be able to buy only newspapers, and only as many as she can find vendors.

Although not suitable as a general replacement for credit cards or cash, such a scheme has a number of important properties that make it especially well suited to the kinds of occasional transactions for which credit cards and specialized digital cash systems are too expensive.

2 Architecture

First, some terminology. The main players in our scheme are *Merchants*, who sell things and collect payments, and *Payers,* who buy things and pay for them. Merchants and Payers sign up for service with a *Provisioning Agent* (PA). Merchants interact with Payers through the *Merchant Payment Processor* (MPP). Payers are assumed to hold portable lightweight devices capable of some processing (cellphones, PDAs, *etc.*) A *Clearing and Settlement Center* (CSC) for reconciling transactions may be a separate entity, or may be part of the PA.

Our microbilling architecture is designed to operate efficiently under a number of constraints.

Foremost is that communication between the PA and the Payers and Merchants is relatively expensive. This implies that transactions must be able to be consummated between Payers and Merchants directly, with the CSC verifying them at a later time[1]. Thus, the Payer will have to provide proof that she is allowed to perform a transaction. Likewise, the Merchant will have to convince the Payer that she really is an authorized Merchant. Assuming a large number of Payers and Merchants, with a potentially high turnover rate, massive periodic reconfiguration of all devices involved is impractical.

A public-key credential-based architecture fits nicely here. The PA would act as a trusted third party to Payers and Merchants, who would be able to

[1] There exist established and efficient infrastructures that can process large volumes of micro-transactions and reconcile these against the associated accounts. Obvious examples are telephone and utility companies, banks, *etc.*

authenticate each other offline using the appropriate credentials. Although computationally intensive, public key operations are not prohibitively expensive on the latest generation of lightweight computing devices (see Section 3 for some performance figures for the Palm PDA). Some devices also provide hardware cryptographic acceleration in the form of ASIC modules. Modern cellphones, for example, already have significant cryptographic support in hardware. These trends should mitigate, if not altogether eliminate, performance concerns.

Loss of the Payer's (or Merchant's) device/credentials should not be catastrophic. While the Merchant, Payer, or PA (depending on agreements) may have to incur some costs as a result of the loss (similar to credit card loss or theft), it should be possible to limit the potential damage. Payer and Merchant credentials utilized should therefore be short-lived (and thus frequently-refreshed) and fairly restrictive with regard to the Payer's or Merchant's capabilities (what items can be bought/sold, in what quantities/prices, *etc.*) Thus, we must be able to encode risk-management strategies in the credentials we use. The refresh rate of the credentials and the capabilities expressed therein depend on the risk-management strategy of the PA and thus have to be flexible[2].

An added benefit of such an approach is that the different levels of physical security and tamper-resistance (ranging from none to very secure) can be taken into consideration on an individual-user basis as part of the PA's risk-management strategy. Other parameters that may influence risk-management can also be adjusted on a per-user basis as well.

The user devices used for the transactions can be very versatile (*e.g.*, cellphones, PDAs, even laptop or desktop computers), and the architecture does not depend on any particular communication technology for the transactions, only that the user device has some ability to communicate back to the PA from time-to-time, whether by telephone, cellular text message, Internet, *etc.*

Since the PA has no physical control over the user device, it is necessary to verify any information received by it. Thus the Payer device must provide prove to the Merchant that it is authorized to perform a transaction, and must able to provide signed messages that the Merchant can use to clear payment for a transaction.

A central principle behind our system is the use of a trust-management system as the core component to express and encode the risk-management strategies. The following subsections give an overview of trust management, describe the mapping between risk and trust management, present our architecture, and discuss how it meets the requirements for an offline micropayment system.

2.1 Trust Management

Trust management, introduced in the PolicyMaker system [BFL96], is a unified approach to specifying and interpreting security policies, credentials, and relationships between users of the system (principals); it allows direct authorization of security-critical actions. A trust-management system provides standard,

[2] The refresh rate also depends on the frequency of communication between the user's device and the PA. We discuss this later in this section.

general-purpose mechanisms for specifying application security policies and credentials. Trust-management credentials describe a specific delegation of trust and subsume the role of public key certificates; unlike traditional certificates, which bind keys to names, credentials can bind keys directly to the authorization to perform specific tasks.

A trust-management system has five basic components:

- A language for describing 'actions', which are operations with security consequences that are to be controlled by the system.
- A mechanism for identifying 'principals', which are entities that can be authorized to perform actions.
- A language for specifying application 'policies', which govern the actions that principals are authorized to perform.
- A language for specifying 'credentials', which allow principals to delegate authorization to other principals.
- A 'compliance checker', which provides a service to applications for determining how an action requested by principals should be handled, given a policy and a set of credentials.

Trust management unifies the notions of security policy, credentials, access control, and authorization. An application that uses a trust-management system can simply ask the compliance checker whether a requested action should be allowed. Furthermore, policies and credentials are written in standard languages that are shared by all trust-managed applications; the security configuration mechanism for one application carries exactly the same syntactic and semantic structure as that of another, even when the semantics of the applications themselves are quite different.

2.2 Mapping Risk Management to Trust Management

Given a trust-management system, it is possible to describe risk-management strategies in the language used for specifying credentials. The details of the transaction (such as item purchased, price, quantity, *etc.*), potential history information (prior transactions between the Payer and Merchant), and other information (time of day, device status, *etc.*), are encoded in the trust management action language. The credential language then determines whether the transaction should be permitted based on that information. The principals in the system (Merchants, Payers, PAs, and CSCs) are identified by their public keys.

User policies identify the PAs that are trusted to introduce other users. For a Payer and a Merchant to be able to perform a transaction, they must share at least one common PA. Note that it is not necessary to restrict the architecture to a two-layer scheme (PAs and Payers/Merchants); multiple layers of PAs can be used.

All user devices and CSCs utilize compliance checkers; these are used by the users to verify transactions as they are performed, and by the CSCs during reconciliation. This allows for a decentralized decision-making process with regards to risk-management, relieving CSCs or PAs from the burden of maintaining a large, online, highly-available infrastructure for transaction verification.

2.3 Microbilling through Risk Management

We can now describe our microbilling architecture in some detail.

The users (Payers and Merchants) are assumed to possess some device that can perform transactions on their behalf. Each user has a public/private key pair, and signs up for service with one or more PAs who issue the necessary credentials. While the details of the business arrangements that might arise are outside the scope of our architecture, it is important to note that these do affect the risk decisions made by the agents and thus the strategies encoded in the credentials. For example, an account with a higher monthly premium may be allowed to make more purchases every day than one with no premium. We give some example credentials in Section 3.

As we have already mentioned, credentials are short-lived to restrict potential damage from loss of the device or compromise of the cryptographic key, and to avoid maintaining and distributing revocation lists at update time[3]. Thus, relatively frequent updates of the credentials have to be performed, depending on the specific implementation. When and how the updates are performed (and their frequency) is device-specific: cellphones might receive their credentials every night in the form of an SMS (or similar) message, or they might place a call to a voicemail or other service number; PDAs might download new credentials every few days during backup. To avoid disruption of service, a device could be issued several credentials representing different tradeoffs between purchasing power and validity duration; the higher-value credential would expire first, but the longer-lived lower value credential would still allow some purchases to be made in the event a credential update is missed. Also note that credentials issued to some Merchants can be made longer-lived, to avoid frequent updates to unattended selling points.

A pair of Payers and Merchants thus equipped with their respective credentials can perform a transaction through a simple authentication/authorization protocol, an example of which is given in Section 3. In simple terms, the two users authenticate each other and then verify each other's capabilities: the Payer verifies that the Merchant is known to the PA and is authorized to charge the Payer's account for the particular type of transaction; the Merchant verifies that the Payer is authorized by a recognized and accepted PA to proceed with the specific transaction.

When a transaction completes, the Payer receives the goods or services purchased; in return, the Merchant receives from the Payer a *Microcheck*, which is a specially encoded signed message that authorizes a one-time charge to the Payer's account and a credit to the Merchant's account; for simplicity, the Microcheck can also be encoded as a trust-management credential.

Periodically, the Merchant provides its collected Microchecks (along with the related transaction records) to the CSC, which uses this information to verify the transaction and charge/credit the relevant accounts. The Payer's device (the

[3] Since we cannot do online revocation check at the time a transaction is underway, the only other approach involving revocation lists involves updating all the users of a PA with the revoked credentials at update time.

"Microcheck Writer") may also keep a record of all transactions, which can be used to reconcile posted and billed charges against the payer's records. CSCs communicate with PAs to indicate the status of Payers' and Merchants' accounts.

Note that it is not strictly necessary for the Payer to authenticate the Merchant during the transaction. Surprisingly, this carries no loss in security: regardless of whether the transaction protocol was completed or not, the Merchant can always not dispense the goods, or the Payer can always claim that she never received them. Mutual authentication does not help in either case, and the dispute would have to be settled through other means[4]. Given the low value of the transactions, and the fact that a dispute history of the Merchant and the Payer can be maintained and referenced by the PA and/or CSC, fraudulent transaction disputes need not be a major concern. For the remainder of this paper, we will assume Payers do not authenticate the Merchants.

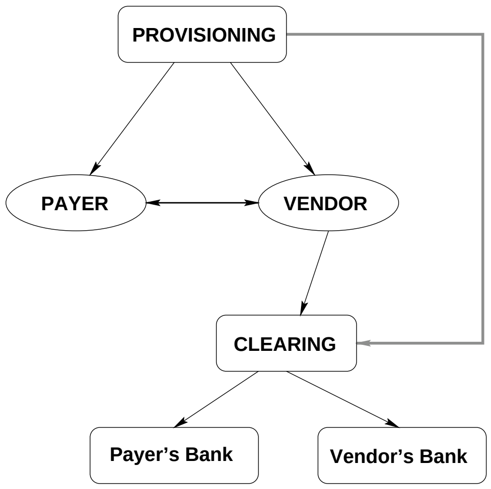


Fig. 1. Microbilling architecture diagram. The arrows represent communication between the two parties: Provisioning issues credentials to Payers and Merchants; these communicate to complete transactions; Merchants send transaction information to Clearing which verifies the transaction and posts the necessary credits/charges or arranges money transfers. Provisioning and Clearing exchange information on the status of Payer and Merchant accounts.

Figure 1 gives a schematic description of the architecture.

[4] The Payer can simply verify the signature on the Merchant credential, to avoid beginning a transaction with an unauthorized Merchant. A fraudulent Merchant cannot use another's credentials, as those will not be accepted by the CSC (and since no charges will be posted to the Payer's account, no dispute needs to be resolved.)

2.4 Security Analysis

We need to examine a number of issues associated with our system. Minimally, we have three types of communication: provisioning, reconciliation, and transaction. We shall not worry about the value transfers to banks, as there already exist well-established systems for handling those. The inter-PA and PA-CSC communication is also relatively simple to secure, and can use well-established cryptographic protocols such as IPsec or SSL.

The Payer must be provisioned over a secure link; security can be physical, as in our prototype where the Palm PDA is connected to the provisioning computer over a direct serial link in the user's presence. More commonly, security would be provided over a cryptographically protected network link. For that purpose, the Payer and the PA must share some long-term security parameters, so that short-term credentials can be generated with fresh public keys every refreshing period. An IPsec or SSL link between the PA and the Payer, or the encrypted channel already being used by a cellphone to exchange signaling information with its provider, is sufficient – there is no need to invent another protocol.

The length of the asymmetric keys generated for the short-term credentials needs some examination. The public key of the Payer may be exposed to third parties during the actual transaction and during the deposit of all the Microchecks by the Merchant. In addition, if the adversary is the Merchant himself, then he already has the Payer's public key. Either way, if an adversary can factor the RSA key of the Payer, all it can do is make purchases for the duration of the short-lived credential. In other words, cryptanalysis of the Payer key is no worse than discovering the secret by stealing the device or hardware compromise. The utility of this to the adversary is limited by the fact that the possible transactions are all low-value ones, and therefore the amortized cost of a machine big enough to discover keys within the time-frame allowed is uneconomical even for relatively small RSA modulus sizes (*e.g.*, 512 bits).

If the Merchant is considered an adversary, the situation is slightly different; the Merchant really also has the extra time between the expiration of the credential and the next time he has to call up the CSC to deposit the Microchecks (since he can "pre-date" fraudulent transactions). Since this might happen only infrequently (perhaps only once a month), the vulnerability interval could be larger. Therefore, it may be advisable to encode a maximum number of purchases per day per vendor ID in the payer's short-term credential. This is where fraud-detection techniques also come into play. The CSC can verify, *e.g.*, that only a reasonable number of purchases have been made per vending machine; it can also verify that the purchases have not been made at unreasonably disparate geographical locations, and so on. This is one of the ways where using a trust-management system with programmable policies (such as KeyNote) is especially advantageous. If better policies for fraud detection have to be deployed, or different policies need to apply to different people, they can all be encoded in the credentials.

Provisioning the Merchant's Payment Processor (MPP) is in many ways easier than provisioning the Payer. The MPP need never sign anything – it only verifies the signatures of the credentials that the Payer sends it. The MPP does need to have long-term secrets so that its communication with the CSC can be secure, but this is just part of the communication protocol, so all the well-known IPsec considerations apply. Provisioning the MPP can be as simple as inserting a floppy disk or a read-only memory chipcard with the necessary IPsec key material. The MPP should also have some concept of real time, but that does not need to be very accurate. In the case of the Merchant, the adversary is the Payer.

Since value is not stored in either the Payer or the Merchant, there is no need for tamper resistance against the owner of the Payer or Merchant hardware. The security of the payment system really only depends on the signing key of the PA; the PA secret must be well protected, and in many applications it will be appropriate to use techniques such as hardware security and shared secrets to protect it.

3 KeyNote Microchecks

It was straightforward to design a practical payment system based on the architecture of the previous section, using the *KeyNote Trust Management System* [BFIK99] as the basis for specifying credentials and risk-management policies.

KeyNote is a simple trust management system and language developed to support a variety of applications. Although it is beyond the scope of this paper to give a complete tutorial or reference on KeyNote syntax and semantics (for which the reader is referred to [BFIK99]), we review a few basic concepts to give the reader a taste of what is going on.

The basic service provided by the KeyNote system is *compliance checking;* that is, checking whether a proposed *action* conforms to local *policy.* Actions in KeyNote are specified as a set of name-value pairs, called an *Action Attribute Set.* Policies are written in the KeyNote *assertion language* and either accept or reject action attribute sets presented to it. Policies can be broken up and distributed via *credentials*, which are signed assertions that can be set over a network and to which a local policy can defer in making its decisions. The credential mechanism allows for complex graphs of trust, in which credentials signed by several entities are considered when authorizing actions.

In our micropayment system, various players issue KeyNote credentials to encode the short-lived risk management strategies and use KeyNote compliance checkers to make risk management decisions. We also use KeyNote credentials to encode the payment messages, which we call *KeyNote Microchecks.*

Let us now examine how we encode various aspects of our system with KeyNote.

3.1 Merchant Policy

Each merchant must have a policy that identifies the public keys of the Provisioning Agents (PAs) that are trusted to issue Payer credentials.

For the purpose of simplicity, we assume that there is only one PA (and only one PA key). Each merchant would then have a KeyNote policy as follows:

```
Local-Constants: PA_KEY = "rsa-base64:MIGJAoGBAM8ibp27l02IIZA+\
      5xANbFmgRtV3Yh0pSic2wk8YB/dGpHQDysmQ9buUtf7pJ/xhW5s+GV\
      4K5HwXsPo1MSim0w4z5fjvCDEfSwzB0fsp7p01u+NWwJyd8hrb/iLYq\
      6tGmhha7R0+KG+fUEvLhArtyV0pQ0oWfVBji4o0tIa9GrGzAgMBAAE="
Authorizer: "POLICY"
Licensees: PA_KEY
Conditions: app_domain == "deli" -> "true";
```

This KeyNote policy essentially says that any actions in the “deli” application should be authorized if they are signed with (or authorized by a credential issued by) the RSA key identified as “PA_KEY”.

This policy is stored in each Merchant’s computer and is consulted whenever an offline purchase is to be made.

As another example from a different application area, the following policy could be used by a car rental agency. This policy not only requires proof of payment, but also a driver’s certificate from the Department of Motor Vehicles.

```
Local-Constants: PA_KEY = "rsa-base64:MIGJAoGBAM8ibp27l02IIZA+\
      5xANbFmgRtV3Yh0pSic2wk8YB/dGpHQDysmQ9buUtf7pJ/xhW5s+GV\
      4K5HwXsPo1MSim0w4z5fjvCDEfSwzB0fsp7p01u+NWwJyd8hrb/iLYq\
      6tGmhha7R0+KG+fUEvLhArtyV0pQ0oWfVBji4o0tIa9GrGzAgMBAAE="
      DMV_KEY = "rsa-base64:MCgCIQGB0f8lSVZfHDwdck\
      ESR/Dh+0NPMrYvd0QlU9QdKbKbRQIDAQAB"
Authorizer: "POLICY"
Licensees: PA_KEY && DMV_KEY
Conditions: app_domain == "car rental" -> "true";
```

3.2 Payer Credentials

The Payer credentials are where most of the risk management strategy is encoded. These credentials are issued to each payer by the PA at relatively frequent intervals and specify the exact conditions under which an offline payment can be authorized. Recall that the Merchant policies will authorize anything authorized by the PA public key; the Payer credentials, therefore, are signed by the PA key and encode the exact restrictions for a given payer.

Different payers can be allowed to do different things, and this will be reflected in the details of their credentials. Each Payer has her own public key, which is encoded in the credential along with the restrictions. For example:

```
Local-Constants: PA_KEY = "rsa-base64:MIGJAoGBAM8ibp27l02IIZA+\
      K5Hw5xANbFmgRtV3Yh0pSic2wk8YB/dGpHQDysmQ9buUtf7pJ/xhW5s+\
      GV4XsPo1MSimOw4z5fjvCDEfSwzBOfsp7p01u+NWwJyd8hrb/iLYq6tG\
      mhha7R0+KG+fUEvLhArtyV0pQ0oWfVBji4oOtIa9GrGzAgMBAAE="
      PAYER_KEY = "rsa-base64:MCgCIQGB0f81SVZfHDwdck\
      ESR/Dh+ONPMrYvd0Ql1U9QdKbKbRQIDAQAB"
Authorizer: PA_KEY
Licensees: PAYER_KEY
Conditions: app_domain == "deli" && currency == "USD"
      && &amount < 1.51 && date < "20001024" -> "true";
Signature: "sig-rsa-sha1-base64:QU6SZtG9R3IXXAU9vRDBguUp\
      PpFgh8s5OOpbOOKOYMRxbzLfVpLvyyzV16fw9uT4Gkq1ToZAdhZVkF5z\
      uhumHXi2wmgZqzFexpoiitvpXRCuERkZPPK60SikMpzi0IfNkPYLiqSp\
      p7mHrdEAChZpPnBTl2tUGxQBK/17fKVSRPa0="
```

This very simple credential allows the Payer holding the PAYER_KEY to make any offline purchases in the “deli” application for up to 1.51 each (in USD) until the date is “20001024”. Presumably, this credential would have been issued to the payer a day or two before that.

The conditions in the Payer credential can be more complex, of course, if the risk management strategy demands it and if the merchant is able to store and refer to state about recent transactions. For example, we might provide a maximum number of transactions or maximum total value than can be purchased from any one Merchant, or it might require that the time between two transactions at the same merchant be at least some interval.

The payer credential is stored on the Payer’s portable computing device and is transmitted to the Merchant whenever she wants to make a purchase.

3.3 Making A Purchase

When a Payer wants to buy something from a Merchant, the Merchant first encodes the details of the proposed transaction into an *offer* which is transmitted to the Payers computer. (We assume that the parties have some mechanism for negotiating the details of the offer, such as entering keys on a cash register or pushing buttons on a vending machine, and that there is some communication mechanism, such as infrared, between the Merchant’s and Payer’s devices).

The offer is a set of attributes and values that describes the transaction, *e.g.*:

```
merchant = "LEE'S DELI"
currency = USD
product = "CelRay Soda"
date = 20001023
amount = 0.55
app_domain = "deli"
nonce = eb2c3dfc860dde9a
```

This offer is from "Lee's deli" and is for a product called "CelRay Soda" that costs 0.55 in USD.

Observe that the name of the Merchant and the product description are just text strings. These, along with the price of the product, will be displayed on the Payer's device to prompt the user for approval.

If the Payer wishes to proceed, she must issue to the Merchant a KeyNote Microcheck for this offer. The Microchecks are also encoded as KeyNote credentials, that authorize payment for a specific transaction. The Payer creates the following KeyNote credential signed with her RSA key, and sends it, along with her Payer credential, from the PA to the Merchant:

```
Local-Constants: PAYER_KEY = "rsa-base64:Mcg..."
Authorizer: PAYER_KEY
Licensees: "LEE'S DELI"
Conditions: app_domain == "deli" &&
      currency == "USD" && amount == "0.55" &&
      nonce == "eb2c3dfc860dde9a" &&
      date == "20001023" -> "true";
Signature: "sig-rsa-sha1-base64:Qpf..."
```

(Key and signature encodings have been truncated for readability.)

This credential is effectively a check signed by the Payer (the Authorizer), and payable to Lee's Deli (the Licensee). The conditions under which this check is valid are that the payment is for something costing 55 cents, purchased on the 23rd of October 2000, and for the particular nonce given in the Merchant's offer. The nonce maps payments to specific transactions, and prevents double-depositing of Microchecks by the Merchant.

To determine whether he can expect to be paid (and therefore whether to accept the payment), the Merchant sends the action description (the attributes and values in the offer) and the Payer's key along with his policy (that identifies the PA key), the Payer credential (signed by the PA) and the Microchecks credential (signed by the Payer) to his local KeyNote compliance checker. If the compliance checker authorizes the transaction, the Merchant is guaranteed that the PA will allow payment. In the case of the policies, credentials, and transaction details given in the examples, the compliance checker would approve the transaction. The correct linkage among the Merchant's policy, the PA key, the Payer key, and the transaction details follow from KeyNote's semantics.

If the transaction is approved, the Merchant should give the item to the Payer and should store a copy of the Microcheck along with the payer credential and associated offer details for later settlement and payment.

If the transaction is not approved because the limits in the payer credentials have been exceeded, then, depending on their network connectivity, either the Payer or the Merchant can request a transaction-specific credential that can be used to authorize the transaction. Observe that if this is implemented transparently and automatically it provides a continuum between online and offline transactions tuned to the risk and operational conditions.

3.4 Clearing and Settlement

Periodically, the Merchant will 'deposit' the Microchecks (and associated transaction details) he has collected with the Clearing and Settlement Center (CSC). The CSC may or may not be run by the same company as the PA, but it must have the proper authorization to transmit billing and payment records to the PA for the PA's customers. The CSC receives payment records from the various Merchants; these records consist of the Offer, and the KeyNote Microcheck and credential from the payer sent in response to the offer.

In order to verify that a Microcheck is good, the CSC goes through the same procedure as the Merchant did when accepting the Microcheck. If the KeyNote compliance checker approves, the check is accepted. Using her public key as an index the payer's account is debited for the amount of the transaction. Similarly, the Merchant's account is credited for the same amount.

3.5 Prototype Implementation

We built a prototype KeyNote Microcheck system based on soda vending machines and Palm PDA computers. It is described in detail in the Appendix.

4 Discussion and Conclusions

We have demonstrated a simple and, for some applications, practical scheme for offline micropayments without the overhead of either secure hardware or online transaction authorization. Our scheme represents a departure from the usual approach to designing such systems. In particular, we chose to tolerate manageable losses, rather than preventing them, and we made no attempt to provide anonymity.

Risk management has long been a central part of the financial world - it is the basic value-service provided by credit card issuers, loan underwriters, insurers, the financial markets, *etc.* In this respect, it is rather surprising that platforms to support risk management techniques for avoiding online authorization have not previously been applied to electronic micropayment systems. (Indeed, older manual credit card processing protocols often included a “floor limit” on transactions below which it a telephone authorization was not required, although the limit was not specific to the individual cardholder.) Instead, previous eletronic systems have focused on preventing fraud and failure, rather than on managing it. Unfortunately, the prevention mechanisms can be too expensive for micropayments, making a risk management approach especially attractive.

We have described a platform that makes it possible to encode risk management rules for offline micropayments. An obvious future direction for research is in the area of systems that generate and adapt these rules to actual operational conditions. We hope to stimulate work in this direction.

References

[BFIK99] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust Management System Version 2. Internet RFC 2704, September 1999.

[BFL96] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proc. of the 17th Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, Los Alamitos, 1996.

[BGH$^+$95] M. Bellare, J. Garay, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. iKP – A Family of Secure Electronic Payment Protocols. In *Proceedings of the First USENIX Workshop on Electronic Commerce*. USENIX, July 1995.

[BGJY98] M. Bellare, J. Garay, C. Jutla, and M. Yung. VarietyCash: a Multi-Purpose Electronic Payment System. In *Proceedings of the Third USENIX Workshop on Electronic Commerce*. USENIX, September 1998.

[Cha82] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Crypto ’82 Proceedings*. Plenum Press, 1982.

[Cha92] D. Chaum. Achieving Electronic Privacy. *Scientific American*, pages 96–101, August 1992.

[CTS95] B. Cox, D. Tygar, and M. Sirbu. NetBill security and transaction protocol. In *Proceedings of the First USENIX Workshop on Electronic commerce*. USENIX, July 1995.

[DB99] N. Daswani and D. Boneh. Experimenting with Electronic Commerce on the PalmPilot. In *Proceedings of the Third International Conference on Financial Cryptography*, number 1648 in Lecture Notes in Computer Science, pages 1–16. Springer-Verlag, 1999.

[DBGM$^+$98] N. Daswani, D. Boneh, H. Garcia-Molina, S. Ketchpel, and A. Paepcke. SWAPEROO: A Simple Wallet Architecture for Payments, Exchanges, Refunds, and Other Operations. In *Proceedings of the Third USENIX Workshop on Electronic Commerce*. USENIX, September 1998.

[dST98] A. de Solages and J. Traore. An Efficient Fair Off-Line Electronic Cash System with Extensions to Checks and Wallets with Observers. In *Proceedings of the Second International Conference on Financial Cryptography*, number 1465 in Lecture Notes in Computer Science, pages 275–295. Springer-Verlag, 1998.

[EBCY96] D. Eastlake, B. Boesch, S. Crocker, and M. Yesil. CyberCash Credit Card Protocol Version 0.8. Internet RFC 1898, February 1996.

[FB98] E. Foo and C. Boyd. A Payment Scheme Using Vouchers. In *Proceedings of the Second International Conference on Financial Cryptography*, number 1465 in Lecture Notes in Computer Science, pages 103–121. Springer-Verlag, 1998.

[Her98] A. Herzberg. Safeguarding Digital Library Contents. *D-Lib Magazine*, January 1998.

[Hic95] K. Hickman. Secure Socket Library. http://home.netscape.com/security/techbriefs/ssl.html, February 1995.

[HSW96] R. Hauser, M. Steiner, and M. Waidner. Micro-payments based on ikp. In *Proceedings of the 14th Worldwide Congress on Computer and Communication Security Protection*, June 1996.

[HY96] A. Herzberg and H. Yochai. Mini-Pay: Charging per Click on the Web. http://www.hrl.il.ibm.com/mpay/, 1996.

[JY96] C. Jutla and M Yung. Paytree: amortized signature for flexible micropayments. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*. USENIX, 1996.

[Man95] M. S. Manasse. The Millicent protocols for electronic commerce. In *Proceedings of the First USENIX Workshop on Electronic Commerce*. USENIX, July 1995.

[MN94] G. Medvinsky and C. Neuman. NetCash: A design for practical electronic currency on the internet. In *Proceedings of the Second ACM Conference on Computer and Communication Security*, November 1994.

[NM95] C. Neuman and G. Medvinsky. Requirements for network payment: The Netcheque prospective. In *Proceedings of IEEE COMCON*, March 1995.

[PHS98] T. Poutanen, H. Hinton, and M. Stumm. NetCents: A Lightweight Protocol for Secure Micropayments. In *Proceedings of the Third USENIX Workshop on Electronic Commerce*. USENIX, September 1998.

[RS] R. Rivest and A. Shamir. PayWord and MicroMint. *CryptoBytes*, 2(1):7–11.

[SET] Secure Electronic Transactions (SET). http://www.setco.org/.

[Tan95] Lei Tang. A Set of Protocols for MicroPayments in Distributed Systems. In *Proceedings of the First USENIX Workshop on Electronic Commerce*. USENIX, July 1995.

A Prototype Implementation

We built a prototype microbilling system in order to test the feasibility of a such a system, and also study users' reactions. We chose to implement our prototype Payer system, the "Electronic Check Writer (ECW)" on a 3Com Palm-III PDA. A large number of people in our location already have Palm PDAs, which makes it easy to get volunteers to test our system without having to buy or carry additional hardware. (Many modern mobile phones have infrared ports, and would also have been suitable as an ECW platform).

We developed the ECW software for the Palm Computing Platform using the ssl port, gnu utilities under Linux. The Palm-III (as well as all newer models) has an infrared interface, and sufficient processing power to compute an RSA signature with a 257-bit modulus in approximately five seconds, which makes for an acceptable user delay. While a 257-bit modulus can hardly be considered secure, it makes for acceptable user delay while still providing proper authentication.

In order to make this an attractive and realistic demonstration, we used an actual soda vending machine. We purchased and modified a generic vending machine from a mail-order distributor. Figure 2 is a sketch of the machine with the associated hardware. It can dispense eight kinds of 12oz cans of soda. It has eight selection buttons in the front, and eight 24V motors to operate the dispense mechanisms. To use it, we removed and discarded the original controller, the bill collector, and the coin collector, and brought the wires that sense the push-buttons and drive the dispensing motors out to a Z-World PK2275 industrial controller (Figure 3), which is programmed in a variant of C called Dynamic C. The controller is connected via a serial port to a Linux PC.

A Vacuum Fluorescent Display (VFD) is mounted on vending machine, and is used to give prompts and status information to the user (Figure 4). It is connected to the PC using a serial port.

Also connected to the Linux PC is a JetEye IrDA adaptor, which is used to communicate with the Palm unit with the IrDA protocol suite.

The "Provisioning Agent" in our system is a BSD Unix workstation with a palm cradle; users get their initial credentials (and the ECW software) in person but receive their ongoing short-term credentials via electronic mail.

Before the system can be put to work, the Provisioning Agent station must set up keys for itself. This one-time operation is done with a key generation command to the KeyNote toolkit. Specifically, we generate an RSA key pair with a modulus size of 1024 bits. The public and private components are stored in regular text files.

The provisioning agent must make its public key known to all Merchants and clearing agents. In our case, the provisioning agent also serves as the clearing agent; sending the key to the Merchants is simply done by copying the public key to the vending machine's permanent storage.

Provisioning each user PDA consists of generating a short-term certificate, in the form of a signed KeyNote assertion, specifying that its public key can sign electronic checks meeting the conditions expressed in the assertion. As an implementation shortcut, the provisioning agent also generates the PDA's RSA

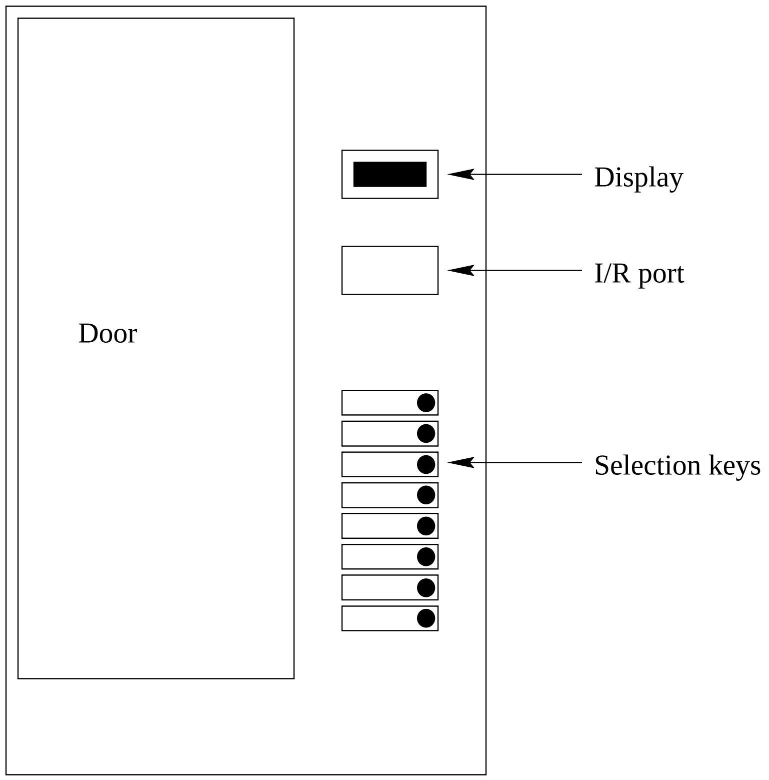


Fig. 2. Front of the vending machine, showing the display, infrared port, and some of the pushbuttons.

key pair, and uses the public key in the assertion it generates. The key pair, along with the short-live certificate, are then compiled into an executable, called “SodaPop”, which is loaded in the PDA.

The PDA is provisioned by going to a provisioning station, which is a PC running Linux, and has a copy of the source code of the ECW application. A simple command line interface is used to invoke a script that builds the new set of keys and the certificate, compiles them into the ECW application, and uploads it to the PDA which has been placed on the synchronization cradle.

Our vending machine looks superficially like a standard vending machine, but the coin mechanism has been replaced with a small display and infrared port. There are eight buttons to select drinks. Upon approaching, the user is prompted to make a selection (see Figure 4, first panel). Hidden inside the machine are the Merchant Linux PC and controller.

When a button is pressed on the soda machine, the supervisor program running on the Merchant Linux PC, which is polling the controller several times a second, registers that a button has been pressed. It then asks the controller

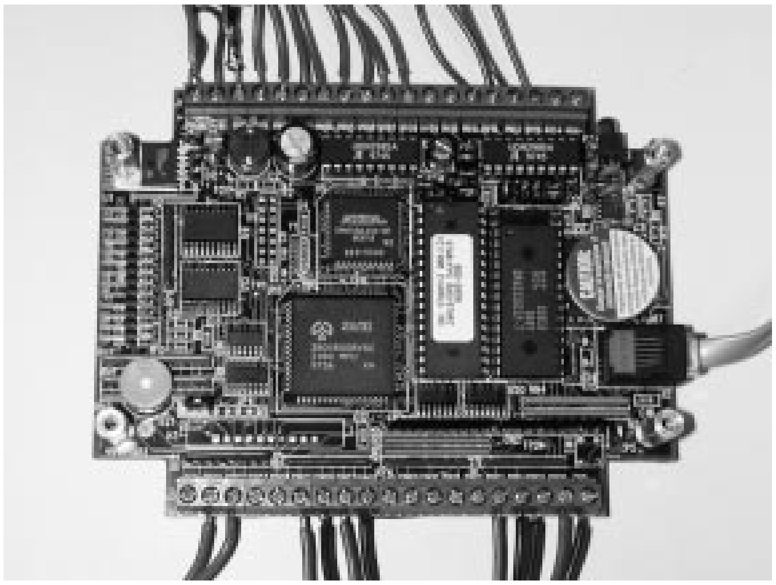


Fig. 3. Z-World PK-2275 Industrial Controller

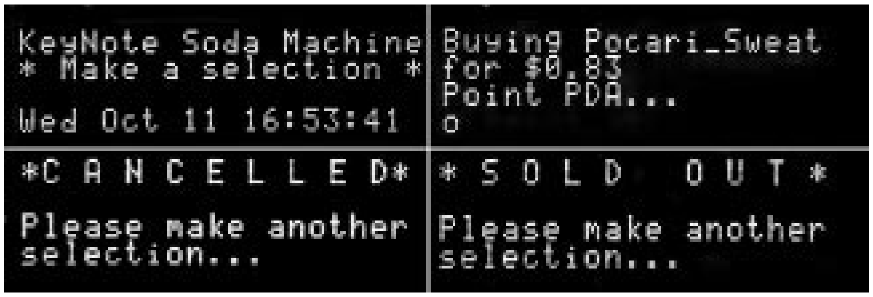
whether the drink is available or sold out (there is circuitry in the vending machine to detect that). If the drink is sold out, the user is told so; if not, the user is told to aim their PDA at the IrDA interface of the vending machine. The PDA is running the "SodaPop" application; figure 5-left shows the opening screen. Communication between the supervisor program and the PDA is established, and an offer is sent to the PDA.

The PDA then prompts the user with the screen shown in Figure 5-right. If the user clicks on "no", no purchase is made (and, as a courtesy to a subsequent user, the PDA sends a "no" to the vending machine so that it will not wait until it times out).

If the purchase is accepted by the user, the PDA will build a KeyNote Microcheck and will send it, along with its credential, to the vending machine via infrared.

Upon receipt, the vending machine will query its local KeyNote compliance checker with the credential and Microchecks it got, along with the provisioning agent's public key, using the fields from the offer as the Keynote Action Attribute Set.

If everything is approved, the PC sends a command to the micro-controller to turn the corresponding dispensing motor on, and, in this example, a nice cold can of Pocari Sweat will be dispensed.



```
KeyNote Soda Machine
* Make a selection *

Wed Oct 11 16:53:41
```

```
Buying Pocari_Sweat
for $0.83
Point PDA...
o
```

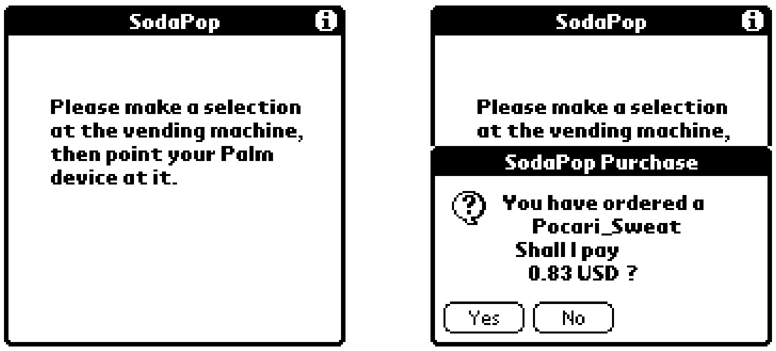
```
*C A N C E L L E D*

Please make another
selection...
```

```
* S O L D   O U T *

Please make another
selection...
```

Fig. 4. Various prompts and status messages.



SodaPop

Please make a selection
at the vending machine,
then point your Palm
device at it.

SodaPop

Please make a selection
at the vending machine,

SodaPop Purchase

You have ordered a
Pocari_Sweat
Shall I pay
0.83 USD ?

Fig. 5. SodaPop PDA Application screens