The following paper was originally published in the
Proceedings of the First USENIX Workshop on Electronic Commerce
New York, New York, July 1995.

The Millicent protocols for electronic commerce

Mark S. Manasse
Systems Research Center
Digital Equipment Corporation

The Millicent protocols for electronic commerce

Mark S. Manasse
Systems Research Center
Digital Equipment Corporation
Palo Alto, California
msm@pa.dec.com
http://www.research.digital.com/SRC/people/Mark_Manasse/bio.html

Introduction

Many protocols have been proposed in the last year which address the problem of securely transferring money over a public network. Most of these schemes have similar goals: to enable transactions with properties similar to those achievable today using credit cards, to provide consumers and vendors with guarantees similar to those afforded by credit cards, or to translate existing payment mechanisms into electronic equivalents. The schemes vary widely in details; some schemes create electronic cash, some schemes work hard to provide anonymity for purchasers, some schemes protect consumers from exposing their underlying accounts to merchants, and some schemes ignore privacy issues altogether.

This is all interesting and necessary, but it fails to address the problems of a market segment that I expect to grow rapidly, once it becomes at all possible: extremely low-priced walk-up transactions. The proposed payment schemes all come with fee schedules that limit transactions to be fairly valuable. In practical terms, for today's systems, and today's mechanisms for transactions, the fees come to a minimum of approximately twenty-five cents for credit-card-like transactions, and at least a penny for services that provide a level of aggregation before charging a credit card. Over time these fees are almost certain to drop, but the current performance of disks, processors, and networks suggest that these fees aren't going to come down due to economic pressures alone: a system that can only handle a dozen transactions per second per computer needs to charge fees in this range in order to be profitable—and that's the kind of transaction rate implied by systems that require non-repudiatable digital signatures. Increased performance over time will increase the transaction rate; I contend that it's interesting to look at the results of applying that to decreasing the price of transactions.

With fees in their current range, the minimum transaction is likely to remain between five cents and one dollar. For tangible goods, this is unlikely to be a problem; even penny candy costs a few cents these days! For information goods, however, the costs for manufacturing and delivery are quite small. If the overhead for billing could be reduced to the same level, what changes might take place in information delivery? Does it matter if the smallest transaction is five cents or a mil?

I contend that it does matter. At five cents, one can decompose a daily newspaper into a few sections (sports, business, world news, local news) without significantly increasing the daily outlay for information. But this level of decomposition doesn't change the nature of information delivery. At a mil, we can decompose the newspaper into individual articles, comic strips, and horoscopes.

And this information marketplace isn't limited to newspapers: individual stock quotes, background stories culled from the morgue, technical papers, historical commercial information, index retrieval, almost anything where the information production can either be automated, or where the market is large enough. An anarchic information provision world, where payment can be rendered not only for the information, but for the filtering to provide just the information that's fit to download.

A decade hence, assuming that computers (and their components) continue on the price and performance curves of the last two decades, the minimum transaction grain will be an order of magnitude smaller than it is today. The smallest transactions will be in the sub-penny range. In a decade, we will see pricing by the web page. We can unbundle the newspaper and the magazine, derive revenue from the newspaper morgue, and sell information without restricting consumers to a subscription to a small number of sources. Citation and reference by hyperlink can replace quotation. “Fair use” photocopying can be replaced by links, and the information gets to the students that care at lower total cost, and with revenue returned to the author rather than the paper and toner companies. Editors, authors, and publishers become more independent, as an editor becomes anyone who can assemble a useful list of pointers to interesting documents, and find a market for that list.

This, I argue, is inevitable. The alternative is not that information remain expensive and bundled, but that copyright become meaningless. Digital storage of information is cheaper than the paper it replaces, and your published works will be photocopied and scanned—as long as it remains overwhelmingly cost-effective to do so.

I have a clear preference between these alternatives. Authors and editors need to be compensated in order to produce quality work. A scheme in which compensation continues to flow to the creators of a piece of work is superior to one in which compensation is incommensurate with use. The remaining question is whether the technology and the will to preserve copyright will leap ahead of the technology and the will to subvert it, or not. I can’t speak to matters of resolve, but I will speak to matters of technology: storage, scanning, and OCR technologies are running ahead of the pricing models for current schemes of electronic commerce. While electronic commerce will eventually realize the kinds of economies necessary to preserve copyright, it’s not certain that it won’t already be too late.

Therefore, it becomes interesting to look at techniques for accelerating the move to sub-penny transactions. There are a few approaches that can be followed.

First, we can look to aggregated service providers. CompuServe, America Online, and (soon) the Microsoft Network provide simple billing models for access to information. The main problem is that the information that they can pay for is information for which they have contracted, and information that they deliver. This makes it hard for smaller information providers to gain access to users, makes it hard for consumers to retain much privacy from their service provider, makes it hard for information providers to determine whether the level of use reported by the service provider is accurate, and makes it hard for consumers to tell whether they are being accurately billed. Do consumers and vendors care about the privacy and accuracy issues? Not so much, today, but a good example of inappropriate behavior by an aggregated service provider might make people care.

Or, we can look at Millicent for an alternative architecture that provides better accuracy guarantees and modest privacy guarantees, if you believe that your payment intermediary isn’t snooping all of your network traffic.

Millicent overview

To achieve privacy, control over expenses, guarantees of payment, and lots of other really good properties, it’s hard to beat cash. A good cheap scheme for digital cash would solve all of the problems right away, except for the problems caused by anonymous transactions, which any FBI or NSA agent will be happy to explain to you.

The problem is, there is no good cheap scheme for digital cash. Any scheme for digital cash is going to boil down to a signed message from a bank that claims to be worth some amount. Checking the signature is going to be relatively easy (although not immediate), but that’s insufficient to tell whether a piece of digital cash is valid. Unlike real cash, duplicating digital cash is cheap and easy; you just send the number a second time to a different merchant. The bank can protect against double spending if the double-spender can be detected and assessed for the amount of duplication; this isn’t going to be enough to prevent largely anonymous users from spending the same electronic coin several thousand times, or from skipping out on their debts. In such cases, the bank would have to deny payment to merchants, or make good on such fraud, driving the transaction price up. To protect against this, merchants could check each coin with the bank, to make sure that it hasn’t been previously spent. This is going to add latency (due to queueing delays) and expense to a transaction system.

One way to make duplicate checking cheaper is to allow everyone to do it. If a local check on a serial number sufficed, we wouldn’t have added much latency to the system. But we can’t do this with cash; the idea of cash is that it’s universal, and thus spendable at multiple vendors simultaneously. The first thing we’ll do in Millicent is to change this.

Let’s define a new kind of currency, which we’ll call *scrip.* Scrip is like cash, in that it has some intrinsic value, but different in that it has that value only when spent with a specific merchant. Like electronic cash, scrip will consist of a signed message attesting that a particular serial number holds a particular value. Unlike typical cash, the message will contain an expiration date, and a particular vendor with whom the scrip can be redeemed.

Since we're concerned with providing transactions in the small, we'll allow ourselves some other simplifications. Ordinarily, digital cash is signed in some way that permits easy verification of the authenticity of a coin. Since a piece of scrip is of value only to its creator, and since the value of a piece of scrip is small, we don't need to be able to prove to anyone other than that vendor that a piece of scrip is authentic. We can assume that vendors are, in some way to be described later, controlled, and that consumers don't buy scrip from vendors that have cheated them before; thus, a consumer's risk in holding scrip from a vendor is limited to the face value of that scrip.

The best analogies to scrip are transit-system fare cards, pre-paid phone cards, and manufacturer coupons. A piece of scrip represents pre-paid value, whose authenticity is of interest only to the vendor, as long as the consumer is confident that it came from a trustworthy source. The vendor can employ whatever technology is appropriate to ensure authenticity and non-duplication.

In the scheme we propose for Millicent, the authenticity is guaranteed by a secret key. Since we don't need to prove the value to anyone other than the single merchant, it suffices to compute a signature incorporating a special secret. We propose that computing MD-5 of the plain text of the scrip followed by a secret key is sufficient, although any one-way hash function will suffice. Encryption would also suffice, but would be slower and might give us more trouble exporting the system.

To avoid duplication, the vendor will keep bit vectors corresponding to subranges of the issued serial numbers for scrip. As scrip expires (or becomes fully spent) the bit vectors can be discarded. This allows us to limit the storage requirements for a piece of scrip to an amortized $1+\varepsilon$ bits. This permits the vendor to keep the database of valid scrip in main memory, speeding up transactions.

This leaves us with one big problem: buying scrip for a particular vendor. We can't buy it directly from the vendor, because that would commit us to large accounts with vendors, tying consumers down to large information providers. To get around this, we postulate scrip brokers: agents who buy contracts from vendors to produce scrip.

Millicent scrip

Scrip in Millicent has to be unforgeable, unusable if copied, and inexpensive to create and validate. By using message digest functions as our primary form of encryption, and using shared-key cryptography when cryptography is necessary, we attempt to satisfy these requirements. A piece of scrip in Millicent consists of several parts. The scrip must identify the vendor, the expiration date, the value, and a small amount of information about the consumer. This information is intended to be limited to just the information necessary to satisfy requirements of age, location of residence (for sales tax and tariff purposes), and information needed to qualify for certain discounts, such as student status. Even so, some of this information may be sensitive for certain consumers; thus, we want to make sure that any portion of the data that the user wants to transmit only in encrypted form can be so transmitted.

For administrative purposes, and purposes of making change easily, it is desirable to create session keys. For this purpose, the first two parts of a piece of scrip are a session id and a session key. The session id serves to remind the vendor how to compute the session key. We propose that the session id contain in it the expiration date and a unique session number; if the vendor has a mapping from ranges of session numbers to secrets, the session key is just the result of signing the session id using the corresponding secret.

Let's introduce some notation. H will be a message-digest function, probably MD-5. SessionID will be the string corresponding to the session id; it includes N, the session number. The vendor remembers secrets X(N), where the mapping depends only on, say, the high-order bits of N, and remembers them as long as the expiration period for all sessions that have those high-order bits lasts.

Then the session key, k, is just H(SessionID^X(N)), where ^ denotes concatenation; this is what we mean by "signing a string (in this case, SessionID) with X(N)".

The remaining session information contains the vendor and consumer information. This information we denote by Info; when transmitting Info, we allow the consumer and vendor to decompose Info into encrypted and unencrypted pieces however it suits them, so long as Info can be recovered intact. When encrypting, the encryption should use k as the key.

Finally, we have to encode a coin. A coin, Coin, is a string containing a value, and a serial number Z. Additionally, we compute a coin key, j, a signature witnessing that Coin belongs to the current session. This signature is the result of computing

H(SessionID^Info^Coin^X(Z)). For coins minted by a broker, we require that Z=N; we strongly recommend that X(Z) equal X(N) only when Z=N. That is, the same secret applies to a range of Z or N; we recommend that Z and N not be chosen in the same range except when equal, and that vendors choose distinct secrets for distinct ranges.

When a consumer wants to spend a piece of scrip, she sends SessionID, Info, Coin, and j to the vendor together with a request Req, and signs SessionID^Info^Coin^j^Req with k. Note that Info and Req may be sent partially encrypted under k; Req may contain instructions requesting that parts of the reply be encrypted. If any change is due to the consumer, the vendor sends new values for Coin and j back to the consumer along with the results of the request; to allow the consumer to check authenticity of the response, the whole response should be signed with k. Note that the new value for Coin will have a fresh serial number, and a (typically) decremented value.

Since H is a cryptographically strong message digest function, no one can create valid scrip without knowing X(Z) and X(N). In particular, no one can determine a session key without X(N), and no consumer can successfully find the matching value for a coin key after modifying SessionID, Info, or Coin. Thus, only parties knowing X(Z) (i.e., the vendor and broker licensed to create scrip for a subrange) can set Info and Coin; no one other than the vendor can change the monetary value of a piece of scrip or tamper with the consumer information. By choosing X(Z) different from X(N), the vendor prevents the broker from upgrading the value of consumer-held scrip; the vendor can now verify that all scrip for which X(Z) is a shared secret has Z=N, and the coin contains the value agreed to in the contract between the vendor and the broker.

Vendors may want to encrypt portions of their replies to requests, to prevent announcing their proprietary data to the world. For data which is "nearly public", that is, for data where the consumer has not requested privacy, it may suffice to encrypt the data with a single key for all users, and sell consumers that key encrypted by the session key.

We propose that for transmission of keys, it may be more efficient, and sufficiently secure, to use H and k as a pseudo one time pad. That is, by choosing and transmitting a random string, t, H(t^k) can be exclusive or-ed with a piece of data of the same length as the result in order to securely restrict transmission of that data to parties that know k. We particularly see an application of this to brokers, who need to transmit values for SessionID, Info, Coin, j, and k to consumers who have just bought scrip; of these, only k and some pieces of Info need to travel securely. The secure pieces of Info can be encrypted under k; the transmission of k must be accomplished securely, or else anyone observing that transmission could spend the scrip. Our current prototype implementation uses the one-time pad technique for encrypting k.

Validation of scrip is straightforward: SessionID is used to recover k, by signing SessionID with X(N); since N is part of SessionID, this can be found. Info, if partially encrypted, is reconstructed using k, and SessionID, Info, Coin, Req and j are signed using k and matched against the transmitted value. A match proves that the sender either knew k, or retransmitted an old request, which would therefore be using a previously spent serial number. Additionally, SessionID, Info, and Coin are checked for tampering by comparing j to the signature using X(Z) of SessionID, Info, and Coin. Finally, Z is checked against the bit vector for freshness; if it is not fresh, this is either a replay, or an attempt by the consumer to respend used scrip.

Money

To make the system work, actual money needs to change hands sometimes, and real signatures are needed to insure that these transfers are authorized and correspond to desired consequences.

To buy vendor scrip from a broker, the consumer needs some broker scrip. The consumer buys broker scrip by using some higher-priced form of electronic commerce (DigiCash, NetBill, the Visa/Microsoft protocol, SSL, whatever), which hopefully provides the ability to transmit k securely. After that, the consumer trades broker scrip for vendor scrip as needed, typically receiving lots of change, and a smaller quantity of vendor scrip. As mentioned above, this vendor scrip will have its own values for SessionID, Info, Coin, j, and k; at least k must be transmitted securely. We anticipate that consumers (or their browsers) will tune the policies of their software to buy enough scrip to cover some modest but anticipated number of transactions with the vendor; quantity discounts from the broker (or high markup on small purchases) will encourage scrip purchasers in this direction. This helps limit the transaction rate seen by brokers to an acceptable level. We anticipate that typical purchases will be in the range of five to ten dollars when buying broker scrip, and approximately one cent when buying vendor scrip.

In order to sell vendor scrip, the broker needs to acquire a license to produce scrip. The license needs to be enforceable through normal business practices; we expect that both pre-paid and consignment sale of scrip licenses will occur, with the choice being made by the broker based on past sales volumes and experience. We expect the development of a standard license for small-time information vendors; we anticipate that this will be a consignment sale, and that there will be a conventional discount to brokers for this scrip.

A license consists of a range of N, the secret X(N) for that range, an expiration period, an agreement as to what information Info should contain, and either a fixed value for each piece of scrip in the range or a maximum total value for all broker-produced scrip in the range and a value bound for individual pieces of scrip. As the system scales, it may become necessary for brokers to refer consumers to other brokers to satisfy scrip requests; in that case, the consumer's broker may offer to establish a small account with the vendor's broker for purposes of completing this transaction.

Rekeying, refreshing, and redemption

In order to protect consumer information requests from the broker, consumers may want to rekey their scrip from a vendor using public-key technology. Vendors can be expected to accommodate this request, at a small price; all that's involved is choosing a new value for N, in a range not licensed to any broker, recomputing j and k, and transmitting the new SessionID, Info, Coin, j, and k, with the new k traveling encrypted. The encryption can either be an exclusive-or against a random string provided to the vendor under the vendor's public key, or encryption under a public key specified by the user. Unfortunately, this last is susceptible to a man-in-the-middle attack by the broker; since the only reason to rekey is to divorce your requests from observation by the broker, this may be inadequate, depending on the threat model we ascribe to our broker. The former has the disadvantage that it places the heavier computational load on the vendor, which doesn't scale as well. However, it is immune to man-in-the-middle attack.

Refreshing is similar, except that there's no requirement that the new k be strongly encrypted; the total value of the remaining scrip is decreasing, so there's less reason to be concerned about chaining keys than is typical in cryptographic protocols. We anticipate that refreshing soon-to-expire scrip will be a low-cost operation; vendors may, in fact, prefer to garbage collect subranges to eliminate the storage of bit vectors, and thus may be willing to refresh for free.

At some point, a browser may determine that certain scrip is unlikely to ever be of use. At this time, the consumer might like to redeem the vendor scrip for broker scrip. As part of a broker's license for vendor scrip, vendors receive a small license for generating broker scrip in order to redeem unused scrip. This scrip can either be spent by the consumer in the normal way, or merged with the consumer's existing broker scrip. It's likely that there will be a stiffer fee for these actions, to limit traffic and encourage careful heuristic design for scrip purchase and return.

Problems

We've seen that consumers can't easily cheat in this system: if H is strong, consumers can't create scrip, they can't alter scrip, and they can't respend scrip. But who's to blame when something does go wrong? Vendors would be well-advised to anticipate that a certain fraction of transactions will get dropped at an awkward moment; the consumer's computer or network connectivity will fail, and the consumer both won't get what she wanted, and won't receive the change owed to her. To ameliorate this, vendors should keep an in-memory log of hash values for recent transactions, and generally allow replay of any transaction, or at least be willing to return the change from that transaction if the information previously delivered is now uselessly stale, as long as the request exactly matches the previous request.

If a consumer apparently tries to respend scrip, there are a few possible reasons:

1) the consumer could be cheating—this is what we're trying to catch.

2) the consumer could be confused—by permitting replay, we solve some of this; by writing browsers that are careful, we avoid a lot of this.

3) the broker could be selling the consumer used or bogus scrip—this is a problem that needs to be monitored, making sure that the vendor keeps track of whether clients of particular brokers have more trouble than others. Consumers who have done nothing wrong should insist that their broker make good on rejected scrip; if the broker won't, find another broker.

4) the vendor could be cheating by reporting valid, unspent scrip as bogus or spent—this is the flip side of problem 3, and one that brokers need to be on the lookout for. We anticipate that most brokers will refund scrip to the consumer, until they develop a suspicion of either the vendor or the consumer. If the former, the broker can warn customers wanting vendor scrip of

apparent problems with the vendor, demand better licensing terms, or stop doing business with the vendor. If the latter, the broker stops doing business with the customer. Consumers can't tell the difference between 3 and 4, except that their broker will be more helpful in case 4.

5) the cryptographic system in use could be insufficiently strong, or the security of the computers involved in producing or holding secrets may be inadequate. Stay vigilant.

Performance

In the scheme outlined above, if no data is encrypted, the cost of a normal purchase is 3 hash functions for validation, and 1 to make change, plus the table lookups. Buying scrip from a broker is similar, plus the scrip generation costs: 2 hash functions to produce j and k, and one more using the suggested one-time pad encryption method for keys. Given current MD-5 speeds, this should permit transaction rates in the multiple hundreds per second. There are on the order of 3*10^8 seconds per year, so we should be able to support around 10^11 transactions per year. We'll divide by a factor of 10 to allow for idle time; this still leaves 10^10 transactions per year (contrastingly, a system involving a single public-key encryption per transaction using 768-bit keys will be lucky to reach 10^9 transactions). If each transaction at a broker is worth 10^-4 dollars (which is one percent of a penny transaction), the gross margin (I think that's the right term) is still 10^6 dollars per year, which should be enough to pay for the loaded cost of one computer. The transactions at vendors are much more profitable, although the load might be lower, and the transaction size is much smaller. Still, even at 10^-3 dollars per transaction, 10^9 transactions pays for the hardware and a person to keep it running; 10^10 transactions makes a lot of money for someone.

Extensions

By adding more information to Coin, vendors can use scrip as a mechanism to deliver coupons: you bought the first page of this article, so here's some scrip that buys you the second page at half price, if you use it in the next hour.

By using the techniques of scrip without brokers, even vendors that prefer a subscription model can prevent subscribers from allowing unrestricted use of accounts by others. Once someone gives away their scrip, they can't use it any more, because the serial number won't match. Thus, at the very least, this solves the problem of someone at a university posting their access code for subscription to a service for all to use, without requiring the imposition of severe rate limiting for subscribers.

As computers get faster, we can extend Millicent-like techniques into lower-priced service domains. At around the time that conventional techniques begin to be able to handle the problems of by-the-page information purchase, we'll be ready to handle the problems of paying for bandwidth by the hop, or paying for electronic mail forwarding. At least, we'll be able to handle these problems for source-routed networks.

Conclusions

The big new idea here is that cryptography can be useful to send yourself a message via an untrusted intermediary. Pushing the cost of storing infrequently-accessed information off to the client allows the servers to run quickly: the network acts as a perfect prefetch unit for loading your memory with the necessary state for the next operation. By way of example, a syndicated horoscope column may have tens of millions of readers; a server that had to store all of this information about subscribers might buckle under the seek time of accessing subscriber records. Pushing it off to clients allows us to have the salient parts of the database arrive in memory precisely when needed, at the small cost of verifying that the information hasn't been tampered with.

The secondary idea is that a service can help aggregate the transactions into a grain large enough to be acceptable to more conventional transaction handlers, without providing complete information about the transaction to the service. By making fund transfer explicit in each retrieval, all parties involved can instantly tell when they're being cheated; by limiting the scale of transaction, consumers can control the level of risk they accept.

Acknowledgments

In spirit, this work clearly owes great debts to lots of previous work in this field. Kerberos was certainly inspirational, as were efforts by DigiCash and First Virtual.