Efficient Commerce Protocols Based on One-Time Pads

Michael A. Schneider and Edward W. Felten
Secure Internet Programming Laboratory
Department of Computer Science
Princeton University
Princeton, NJ 08544 USA

Abstract

We present a new commerce protocol that allows customers and merchants to conduct face-to-face credit-card authorizations with a credit card company securely with the option of anonymity for the customer, the merchant, or both. Our protocol guarantees that both parties agree to and know the outcome of each transaction. Our protocol has three advantages over others. First, we need only two Message Authentication Code (MAC) operations per party per transaction, fewer than most popular protocols. Second, our own MAC function, OTPMAC, does not rely on the existence of one-way functions or on any other unproven hypothesis. Third, our protocol generates a new one-time identifier per party per transaction, preventing the linkage of multiple transactions to a single party. Additionally, the protocol can operate in modes using alternatives to the one-time pad, including cryptographic random number generators, and conventional cryptographic MAC functions.

1 Introduction

Traditional credit-card commerce suffers from a number of vulnerabilities. Customers are identified only by a static identifier – their credit card number. Credit card numbers may be stolen, copied, or even randomly generated. Since credit card numbers are static, a number that is copied or stolen may be used for fraud months or even years later. Fraudulent vendors might charge a customer a different amount than is printed on the customer's receipt. Since a more secure system would most likely introduce complications to the use of credit cards, the vulnerabilities are usually simply accepted by a population of users who are willing to trade security for ease of use.[1]

Our commerce protocol suffers from none of these vulnerabilities. The static identifiers of traditional credit cards have been replaced with *one-time identifiers* that are used for one transaction only. In addition, Message Authentication Codes (MACs) are used to secure transactions against tampering. This combination of one-time identifiers and MACs allows both customers and merchants to conduct transactions with extremely high confidence in the integrity of those transactions, even when they do not learn each other's identity.

In some situations, including the face-to-face transaction environment described below, it may not be necessary to provide dual-anonymity. We believe that our protocol is still good for these situations due to its efficiency and provable security.

Our protocol has advantages over other electronic commerce protocols such as SET[8][12], MilliCent[4][7], and NetBill[2][13]. Only two MAC operations are required per transaction, as compared to three or four hash or signature operations for MilliCent, and an even greater number of RSA operations for SET. Our protocol also provides a significant degree of anonymity, including the inability to link multiple transactions to the same user, not found in these other systems.

Our commerce protocol uses a one-time pad to generate one-time identifiers and MACs. If the one-time pad is chosen by some truly random mechanism, then it is possible to prove that the probability of forging a message is vanishingly small. In particular, the security of the protocol is not dependent on the existence of one-way functions or any other unproven assumption. However, since one-time pads are not always practical, our protocol can use a pseudorandom stream, such as one based on a message authentication code (MAC).

2 Background & Purpose

2.1 Transaction Environment

Our commerce protocol is designed to provide security and a degree of anonymity for both a customer Alice and

[1] In some cases, users are protected by legal or other means, but some party (e.g. the card-issuer) remains liable. In all cases, the liable party is the one who trades their own liability for (the user's) ease of use, so this exception doesn't change the basic argument.

1063-9527/00 $10.00 © 2000 IEEE

a merchant Bob operating in a conventional "checkout line" payment transaction environment. Our protocol is particularly concerned with the authorization phase of a conventional credit card transaction.

Our protocol gives both the Alice and Bob the option to conceal their own identities. When both identities are concealed, both Alice and Bob must externally check that they are receiving messages from the intended party in order to prevent a man-in-the-middle attack. This is easy to do for face-to-face transactions, where the customer and merchant have some kind of physical link. The "Anonymity Options" section below provides some additional solutions.

It is assumed that the customer physically possesses a token (e.g. smart card, digital wallet, etc.) capable of running this protocol. The customer communicates with the merchant, who then connects to a Credit Card Company (CCC) for transaction authorization, processing, and confirmation.

We will assume that the customer and merchant share a common CCC. The protocol is easily extensible to provide for multiple distinct CCCs. These extensions would also provide for the transport and handling of the "brand" information of a conventional credit card system.

2.2 Guarantees

Our protocol guarantees that, with overwhelming probability, the CCC will process no transaction without the awareness and agreement of both the customer and the merchant. In addition, the protocol will reconcile all authorized transactions, so that both the customer and merchant will be independently certain of the outcome of each transaction, also with overwhelming probability.

2.3 Man-in-the-middle Attacks

Although both the Alice and Bob must authenticate themselves to the CCC, it is not necessary for them to reveal their identities to each other. When used in the anonymous mode, this protocol is inherently subject to a "man-in-the-middle" type of attack. (Any fully anonymous protocol is subject to this type of attack.) In the attack, an attacker poses as Bob to Alice by making an identical payment request as Bob's. When Alice authorizes the transaction, she will actually be paying the attacker, instead of Bob. This "merchant spoofing," and the analogous "customer spoofing" attack are only possible if an attacker can alter the first two messages of the protocol, where the parties are first identified. Note that in order to perform a "merchant spoofing" attack, the attacker must also be registered with the CCC as a merchant. Since the real merchant (who expected to participate in the protocol but did not do so) immediately detects the fraud, it can be reported to the CCC, who can easily trace the transaction back to the attacker.

The defense against this attack is to provide some sort of reliable transport between Alice and Bob for the first two messages. Since the protocol will most often be used for face-to-face transactions, we simply require some sort of physical connection between Alice and Bob. For example, the customer should physically attach or insert their token into the merchant's equipment. It is possible to extend the protocol to remove this restriction (e.g. to allow use in Internet transactions). However, we cannot defend against man-in-the-middle attacks without compromising the anonymity property enjoyed by the face-to-face protocol.

2.4 Malicious Customers and Merchants

Our commerce protocol is designed to protect Bob if Alice is a malicious customer who attempts to commit fraud. A malicious customer, of course, is not required to adhere to the protocol, and might supply arbitrary messages to the merchant. Even so, the probability of a successful forgery attempt can be made arbitrarily low.

Our commerce protocol is also designed to protect Alice if Bob is a malicious merchant who attempts to charge Alice more or less than she intended to pay. This includes protection against Bob using stored or memorized information from previous transactions to authorize additional transactions without Alice's knowledge. Alice is also protected against an attack in which Bob says that a successful transaction has actually failed, and then requests that she authorize the transaction again, thus double-charging her.

Finally, both customer and merchant are protected from compromised communication links, where messages between the merchant and CCC are observed, intercepted, altered, or forged. The protocol still guarantees, with overwhelming probability, that only mutually agreed upon transactions will be accepted by the CCC.

3 Players

Alice, Bob, and the CCC must all participate in the protocol. Figure 1 shows the three players, as well as the secrets they must have exchanged in advance.

3.1 Alice, the customer

Alice, the customer, must share have a preexisting secret-sharing relationship with the CCC. The shared secrets include a read-only one-time pad and a sequence number that can be incremented. After the secrets are established, Alice does not need to communicate directly with the CCC.

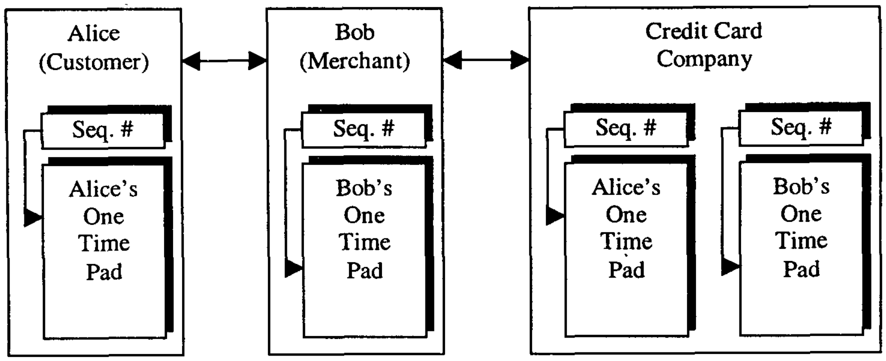

Figure 1: The Players

3.2 Bob, the Merchant

In addition to the physical connection to the customer, Bob, the merchant, is required to establish a connection to the CCC. This connection does not require any additional security above that provided by our commerce protocol.

Bob must also have a preexisting secret-sharing relationship with the CCC. This relationship is identical to the customer/CCC relationship. In fact, either Alice or Bob could act as a customer or merchant in a transaction, since the protocol treats merchants and customers in the same manner, although the CCC will probably only allow customer-merchant transactions.

3.3 The Credit Card Companies (CCCs)

For the purposes of discussion, it is assumed that Alice and Bob share a single CCC. Thus, a single entity is able to verify the identities of both Alice and Bob, as well as their authorization of transactions.

By a simple extension, our commerce protocol may be modified so that customers and merchants have separate CCCs, as long as the CCCs trust one another. In this case, the merchant contacts his own CCC, who verifies the merchant's identity. Then the merchant's CCC contacts the customer's CCC, who verifies the customer's identity. If the CCCs agree, they process the transaction together. Finally, the customer's CCC generates a confirmation for the customer. The merchant's CCC generates a confirmation for the merchant, and relays both over the merchant/CCC link.

Since the negotiation between CCCs is outside the scope of our commerce protocol, and since merchant communication is limited to the merchant's own CCC, the two CCCs can be viewed as a single "composite" CCC, shared between customer and merchant. This is the justification for the simpler model used in our discussion.

4 Protocol Basics

4.1 One-time Identities

To preserve anonymity, both Alice and Bob compute and use one-time identities. These identities are random numbers and are only used for a single transaction. They are the only identifying information required in messages. The one-time identifiers are derived from the one-time pads shared between Alice/Bob and the CCC, allowing the CCC to identify Alice/Bob, but preventing anyone else from determining their identities.

4.2 Anonymity Options

Our protocol allows each party the option of revealing their own identity. Each party can decide independently whether to remain anonymous or reveal their identity. If a party decides to reveal their identity, then they must have agreed on a public identity string with their CCC. This identity string might contain multiple representations of a single identity (e.g. multiple character sets or languages), but the entire string is always transmitted and used in MAC calculations as a unit. This identity string is transmitted with the one-time identifiers between Alice and Bob. Each party who receives a revealed identity incorporates that identity into their respective MAC.

The public identities are not sent to the CCC. The CCC can identify both parties by their one-time identities alone, and can then look up the public identities as necessary. Since the public identity for each party is included in the MAC of the other, the CCC will detect if a party lies about their own identity (the MAC will not check properly).

The man-in-the-middle attack described above is made much more difficult if Bob chooses to reveal his identity. This allows Alice to be sure she is paying the correct

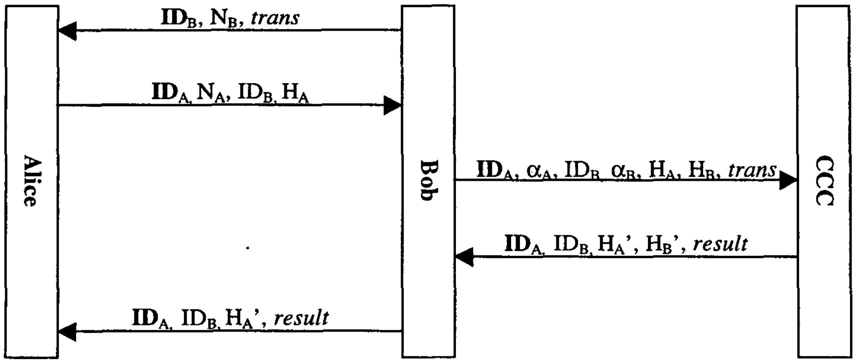
$\mathbf{ID}_B$, N_B, *trans*
$\mathbf{ID}_A$, N_A, $\mathbf{ID}_B$, H_A
Alice
Bob
$\mathbf{ID}_A$, α_A, $\mathbf{ID}_B$, α_B, H_A, H_B, *trans*
CCC
$\mathbf{ID}_A$, $\mathbf{ID}_B$, H_A', H_B', *result*
$\mathbf{ID}_A$, $\mathbf{ID}_B$, H_A', *result*

Figure 2: Normal Protocol Operation

merchant, although it doesn't prevent an attacker from making Alice's payment for her.

4.3 Message MACs

To prevent forgery, Alice and Bob each independently calculate MACs on certain values. These MACs are calculated using OTPMAC and the one-time pads shared between customer/merchant and CCC. This allows the CCC to verify the authenticity of a transaction authorization, and prevents an attacker from forging one.

5 Standard Protocol Messages

A normal transaction in our protocol requires five messages. If synchronization is necessary, two additional messages are used. In addition to the information detailed below, each message begins with a tag identifying the protocol and the message type. The message tags are {M0, M1, M2, M3, M4, M5, M6}.

In preparation for the transaction, Bob must prepare a string *trans*, which contains the actual transaction parameters (e.g. amount of money). After receiving Bob's message, the CCC will generate the string *result*, which gives both Alice and Bob the result of the transaction. The protocol guarantees that both Alice and Bob will accept identical copies of both *trans* and *result*.

When a transaction begins, principals use their own sequence number to index into their one-time pads. The following values are fetched from the one-time pad in order:

- **ID**, used as the party's one-time identifier
- **SYNC**, used in synchronization messages
- **Q**, used to calculate a MAC for the authorization
- **R**, used to verify the MAC for the result

Subscripts represent the values calculated by a particular party. For example, $\mathbf{ID}_A$ represents Alice's value for **ID**. The MAC of a message X using the secret key k is represented by $_{\text{MAC}}(k, X)$.

5.1 Tag M0: Transaction Information

Bob → Alice: {$\mathbf{ID}_B$, $\mathbf{N}_B$, *trans*}

In this message, Bob is asking Alice to authorize *trans*. $\mathbf{N}_B$ is either Bob's real name, or an empty string, if he chooses to be anonymous. In this case, Bob doesn't reveal his own identity to Alice since $\mathbf{ID}_B$ is indistinguishable from a random number.

5.2 Tag M1: Customer Authorization

Alice → Bob: {$\mathbf{ID}_A$, $\mathbf{N}_A$, $\mathbf{ID}_B$, H_A}
$H_A = {}_{\text{MAC}}(Q_A, \{M1, \mathbf{ID}_A, \mathbf{N}_A, \mathbf{ID}_B, \mathbf{N}_B, \textit{trans}\})$

Alice commits to *trans* by computing and sending her one-time identifier and MAC in this message. The presence of $\mathbf{ID}_B$ in this message allows Bob to identify the transaction to which this message refers. $\mathbf{N}_A$ is either Alice's real name, or an empty string, if she chooses to be anonymous. The four fields of this message are inseparable since the MAC has been calculated using $\mathbf{ID}_A$, $\mathbf{N}_A$, and $\mathbf{ID}_B$.

5.3 Tag M2: Merchant Request

Bob → CCC: {$\mathbf{ID}_A$, α_A, $\mathbf{ID}_B$, α_B, H_A, H_B, *trans*}
$H_A = {}_{\text{MAC}}(Q_A, \{M1, \mathbf{ID}_A, \mathbf{N}_A, \mathbf{ID}_B, \mathbf{N}_B, \textit{trans}\})$
$H_B = {}_{\text{MAC}}(Q_B, \{M2, \mathbf{ID}_A, \mathbf{N}_A, \mathbf{ID}_B, \mathbf{N}_B, \textit{trans}\})$

By sending his one-time identifier and MAC with this message, Bob also commits to *trans*. To send the message, Bob must gather both **ID**s, both MACs, and *trans* itself, and send the whole bundle to the CCC.

The flags α_A and α_B are inserted by Bob to tell the CCC if either Alice or Bob (respectively) have chosen to be anonymous. (Bob knows Alice wants to be anonymous when she sends him an empty string.) This information tells the CCC to use an empty string in place of the each party's real name in the MAC calculation. If these flags are altered by any party, the transaction will simply fail, since the MAC will not match when calculated by the CCC.

The CCC looks up $\mathbf{ID}_A$ and $\mathbf{ID}_B$ in its own tables (see the Performance discussion below for more details). It can then retrieve the one-time pads for Alice and Bob, and use these to verify the MACs. Finally, the CCC can perform the transaction.

5.4 Tag M3: CCC Response

CCC → Bob: $\{\mathbf{ID}_A, \mathbf{ID}_B, H_A', H_B', result\}$
$H_A' = {}_{\mathrm{MAC}}(\mathbf{R}_A, \{\mathrm{M3}, \mathbf{ID}_A, \mathbf{ID}_B, result\})$
$H_B' = {}_{\mathrm{MAC}}(\mathbf{R}_B, \{\mathrm{M4}, \mathbf{ID}_A, \mathbf{ID}_B, result\})$

The CCC returns the string *result*, informing both Bob and Alice of the outcome of *trans*. To prevent forgery, the CCC computes MACs for each of Alice and Bob using $\mathbf{R}_A$ and $\mathbf{R}_B$ respectively. These MAC values bind the five message elements together. The **ID** fields allow Bob to determine to which of Bob's transactions this message refers. Bob now has positive confirmation of the transaction outcome. He increments his sequence number, and although he is still required to send M4 on to Alice, he is immediately ready to begin another transaction.

5.5 Tag M4: Response Relay

Bob → Alice: $\{\mathbf{ID}_A, \mathbf{ID}_B, H_A', result\}$
$H_A' = {}_{\mathrm{MAC}}(\mathbf{R}_A, \{\mathrm{M4}, \mathbf{ID}_A, \mathbf{ID}_B, result\})$

This message contains only those portions of message type M3 which are relevant to Alice. Alice performs the same MAC check as Bob to verify the integrity of *result*. Alice can also use the **ID** fields to bind this message to an in-progress transaction. Alice now has positive confirmation of the transaction outcome. She increments her sequence number and is ready for another transaction.

6 Synchronization Messages

Normally, the CCC can pre-compute Alice's next one-time identity using the CCC's own copy of her sequence number and one-time pad. Then, when a message arrives, a simple hash operation or table lookup can be used to identify Alice. Once the transaction is completed, the sequence number is incremented, and the process repeats.

However, should the protocol terminate before completion, the sequence numbers of Alice and the CCC, and thus their pre-computed one-time identities, may not match. This lack of synchronization is a serious problem since the CCC will no longer be able to identify Alice. This might happen if the CCC, merchant, or link goes down in the middle of a transaction, or if the merchant violates the protocol by not sending the last message.

Bob is able to attack Alice by claiming that the CCC is unreachable while actually processing the transaction (taking Alice's money) and withholding the last message (presumably also withholding the goods Alice intended to purchase). Since Alice has no direct communication with the CCC, she cannot immediately detect this attack. However, she will notice it later, and can dispute it like she would dispute any inaccurate transaction.

6.1 Suspicious State

To solve this synchronization problem, one additional state is added to the protocol. If Alice commits to a transaction (by sending M1), but does not receive a valid confirmation message M4, then she is said to be in the "suspicious" state. When Alice is suspicious, Alice's own sequence number might be one less than the CCC's idea of Alice's sequence number. Alice does not engage in any additional transactions until she leaves this state by the synchronization protocol below.

This synchronization protocol might be initiated by Alice while she is still connected to the merchant, or it might be initiated when Alice reaches another merchant. This protocol must complete before any additional transactions are permitted, so the new merchant acts as a courier for the synchronization messages. Since Alice presumably wants to make a purchase when she presents her token to the next merchant, there is an incentive for merchants to provide the synchronization courier service.

6.2 Tag M5: Synchronization Request

Alice → CCC: $\{\mathbf{ID}_A, \mathbf{SYNC}_A\}$

Upon receipt of message M5, the CCC looks up the one-time identifier. Since the identifier in M5 might have been already used by the customer, both the current and immediately previous one-time identifiers must be pre-computed by the CCC. (Matching against this twice-as-large database of identities is only necessary for synchronization messages.) If the one-time identifier has been used on a successful transaction, the CCC resends the original M4 response. If the one-time identifier has not been used, then the CCC generates message M6, and increments Alice's sequence number.

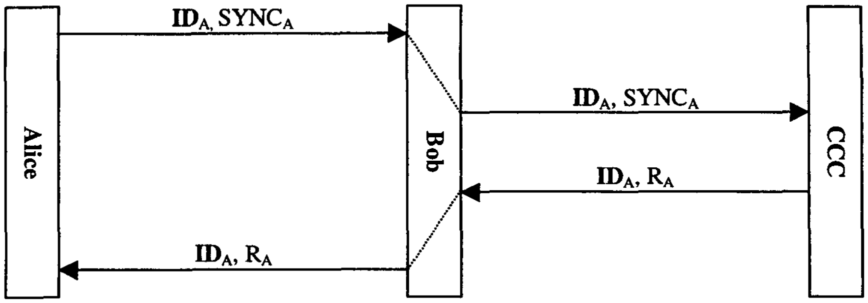

Figure 3: Synchronization Process

6.3 Tag M6: Synchronization Response

Alice → CCC: $\{\mathbf{ID}_A, \mathbf{R}_{A\,[\text{see text}]}\}$

M6 is only returned if no transaction used the one-time identifier specified in M5. In this case, the CCC has received the current (suspect) one-time identifier for the customer, and incremented the sequence number to the next one-time identifier. This means that any transaction authorized by the customer using the suspect identifier is effectively void. This prevents a merchant or attacker from delaying a transaction and resubmitting it later – once the synchronization protocol completes, the delayed transaction is no longer valid.

The field $\mathbf{R}_A$ differs from previous fields since it exposes bits that are, at one point in the protocol, used as the secret in a MAC calculation. However, since M6 is only returned when the CCC never processed a transaction, M6 will only be sent when these bits have not been used. $\mathbf{R}_A$ is a very long value, but only the last few bytes (the same size as a **SYNC** field) should be included. This helps prevents CCC response messages from being forged. After synchronization, the customer is ready to begin a new transaction.

7 Operational Modes

The generation of one-time identities and MACs can be done in various ways, depending upon the operational requirements. The following modes exhibit various capabilities of the commerce protocol, and may be appropriate in different situations.

7.1 OTP (Unconditionally Secure) Mode

In OTP mode, the secrets shared between client and CCC include a truly random one-time pad. OTPMAC is used to generate MACs for messages. This mode is unconditionally secure. It does not rely on the existence of one-way functions, or on any other unproven conjecture. Given any value for the security parameter σ, it can be proven that the probability of forging just one message is less than σ for suitably chosen parameters.

This mode requires that both the client and CCC store a (possibly large) stream of random bits. In addition, once the one-time pad is exhausted, no additional transactions may take place.

7.2 CPRNG Mode

In the case where the storage of a large one-time pad is not practical, pseudo-random numbers can be used instead. In this mode, a cryptographic pseudo-random number generator[3] (CPRNG) is used to generate the bits of the one-time pad as necessary; all other operations are identical to OTP mode. Only a seed for the CPRNG needs to be shared between client and CCC. In this mode, the user can continue engaging in transactions forever – there is no one-time pad that can be exhausted. However, the security of the protocol is bounded by the cryptographic strength of the CPRNG used.

7.3 HMAC Mode

The HMAC mode is similar to the CPRNG mode, but uses HMAC[1][5] in two distinct ways. First, HMAC is used as the CPRNG to generate one-time identifiers, as in CPRNG mode. Second, HMAC replaces OTPMAC as the MAC function. This mode would be most useful when a cryptographic hash function can be calculated more quickly or easily than OTPMAC (perhaps in a hardware implementation), and when we are not worried about the lack of proven security for cryptographic hash functions. There is no one-time pad to exhaust in HMAC mode, so the client may continue engaging in transactions for a very long time. A secret key and sequence number are shared between client and CCC. (Of course, periodic key changes are always prudent.)

To compute a one-time identity, an HMAC operation is applied to the sequence number, using the secret key.

This generates an unpredictable identifier, which does not reveal the secret key. The SYNC field is generated in a similar way, although this calculation may be delayed since it is not needed in most transactions. The HMAC mode of our prototype implementation uses 160-bit one-time identifiers generated by HMAC-SHA-1[6][10].

7.4 Hybrid Modes

The implementation difference between OTP and CPRNG modes is that OTP mode stores a one-time pad, while CPRNG mode generates the values as needed. This means that OTP-mode tends to use more space, while CPRNG-mode tends to use more time. These modes can interoperate in a hybrid mode for the benefit of both customer and CCC.

In hybrid-mode, one party uses a CPRNG to generate a one-time pad for the other party. The second party uses the pad as in OTP mode. The first party stores only the CPRNG, not the precomputed values, and uses it to regenerate the values as necessary as in CPRNG-mode.

For example, suppose the customer uses a device with extremely limited computational power, but an acceptable amount of storage. While the CCC uses a device with enormous computational power, it is expensive to store large one-time pads for a large number of clients. A hybrid mode is best suited to this situation; it reduces client computation, while also reducing CCC data storage requirements. This mode can also be used if the CCC uses a secret generator function, since the CPRNG need not be disclosed to the clients, and is not extractable from reverse engineering of the client's token or software. As in OTP mode, the client will eventually exhaust the supply of values from the one-time pad. Also, as in CPRNG mode, the protocol is as secure as the CPRNG.

8 OTPMAC

8.1 Background

OTPMAC is a MAC function specifically tailored for the calculation of MACs using one-time pads instead of secret keys. In contrast to most MAC functions, OTPMAC does not rely on the existence of one-way-functions, or on any other unproven hypothesis. The probability of an attacker successfully violating the integrity of a message can be bounded below a security parameter σ, for any chosen value of σ.

8.2 Choosing a prime p

OTPMAC uses a large prime number p; it is convenient to choose p as a Mersenne prime (a prime of the form 2^b-1). To satisfy the security criterion, p should be greater than $1/\sigma$. The two primary operations in the calculation of OTPMAC are addition and multiplication mod p. Given a true random one-time pad, OTPMAC returns a uniformly randomly distributed number mod p.

8.3 Calculation Steps

The first step of OTPMAC is to break the message into chunks $\{C_0, C_1, ..., C_{n-1}\}$. Each chunk should correspond to a number less than p, so less than b bits at a time are taken from the message to form each chunk. The last chunk is padded with zeros to a convenient boundary. Finally, one additional chunk C_n containing the length of the original message in bits (plus one) is appended.

Next, a key K_i is generated for each C_i. K_i is a uniformly random value mod p. Exactly b bits are taken from the one-time pad for each K_i. If the value is equal to p (i.e. if all bits are ones) then the value is discarded and the process is repeated. This process guarantees that the keys are uniformly randomly distributed mod p.

The next step is to multiply (mod p) each C_i by the corresponding K_i. All of the resulting products are summed (mod p) to obtain the final OTPMAC value.

8.4 Desired Property

A good MAC function should have the following property: If an attacker has seen a stream of messages and their MACs, but does not know the keys, the attacker should not be able to generate a correct MAC (with a probability greater than that of random guessing) for any messages other than the ones the attacker saw.

8.5 A Useful Lemma

The product (mod p) of any nonzero constant and a uniform random variable (mod p) is a uniform random variable (mod p). Also, the sum (mod p) of any constant (or independent variable) and a uniform random variable (mod p) is a uniform random variable (mod p). Thus, any linear function (mod p) on uniform random variables (mod p) with at least one nonzero coefficient (mod p) is a uniform random variable (mod p). This lemma is useful in showing that OTPMAC does have the desired property.

8.6 OTPMAC Property

The result of an OTPMAC calculation is a linear function on the uniform random keys K_i. The coefficients of the linear function are the message chunks C_i, and the length of the message plus one. The result of an OTPMAC calculation is:

$$C_0K_0 + C_1K_1 + \ldots + C_{n-1}K_{n-1} + (n+1)K_n$$

We will demonstrate that any change to a message will induce a change in the OTPMAC result. In each case

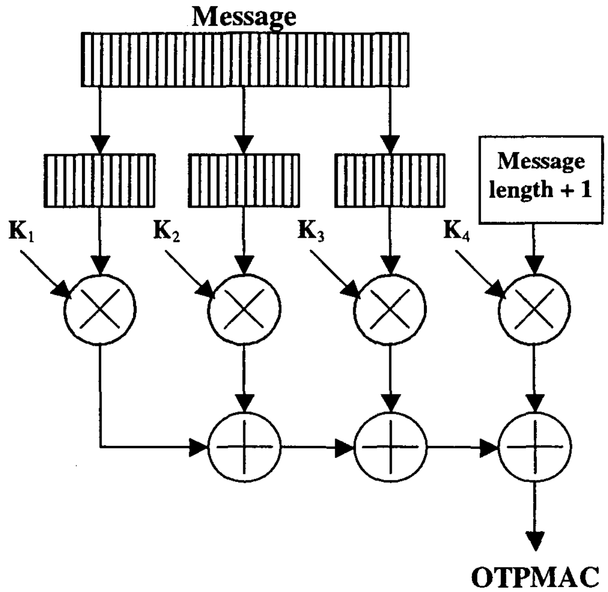

Figure 4: Calculation of OTPMAC

below, the change will be a linear function on the K_i, with at least one nonzero coefficient, and with linearly independent coefficients from the coefficients corresponding to the original message. This property guarantees that an attacker, without knowledge of the K_i, cannot calculate the OTPMAC for the changed message with any probability greater than random guessing. Since the probability of a correct guess is exactly $1/p$, and $\sigma \geq 1/p$, the OTPMAC check will succeed at the receiver with probability no greater than σ.

For example, suppose an attacker changes some C_i to C_i'. This causes OTPMAC to change by the amount $\Delta = K_i(C_i' - C_i) \pmod{p}$. $(C_i' - C_i)$ must be nonzero, so the difference Δ is a linear function with at least one nonzero coefficient on a uniformly random variable. The result of such a function is a uniformly random variable.

The same is true if the attacker attempts to modify more that one chunk of the message. Each C_i which is changed to a different value C_i' forces another coefficient of Δ to be nonzero. As long as there is at least one nonzero coefficient, Δ will be a uniform random variable.

In these arguments, we have assumed that the coefficients of the changed message are linearly independent from the coefficients generated by the original message. However, since (n+1) is always a coefficient, an attacker can't pick any set of C_i that are not linearly independent. Since OTPMAC requires that the K_i be used only once, an attacker only gets one value generated using a particular set of K_i. This prevents the attacker from picking a message to be a linear combination of two other messages – the only linearly dependent messages must be simple multiples of the original message. However, since the length (n+1) is not changed, any change in any coefficient will result in a linearly independent set of coefficients. Therefore, our assumption is justified.

Next, suppose an attacker modifies a message by changing its length from n to n'. If the new length is shorter than the old length by one, then Δ is $(n-C_{n-1})K_{n-1} - (n+1)K_n$. Since (n+1) is nonzero, Δ is again a linear function on uniform random variables, and thus a random variable. This argument can be extended to include all messages shorter than n. Δ will always contain the term $(n+1)K_n$, and will never have any other terms containing K_n. Since (n+1) is always nonzero, the resulting Δ will always be a uniform random variable.

Finally, suppose an attacker lengthens the message. In this case, Δ will always contain a term $(n'+1)K_{n'}$. (n'+1) is again nonzero, so Δ will be a linear function on uniform random variables, and thus a random variable.

Since the OTPMAC value cannot be calculated without the keys K_i, an attacker cannot successfully forge a message with probability greater than σ.

8.7 OTPMAC Conclusions

- OTPMAC works well for the calculation of MAC values when a sufficient supply of random bits is present in the form of a synchronized one-time pad.

- Since the one-time pad is required for both generation and checking of MAC values, OTPMAC can only be used in symmetric key systems.
- OTPMAC can be used to provide virtually any degree of security with suitable choices of σ and p.
- OTPMAC can provide guarantees about the probability of forgery since it does not depend upon the existence of one-way functions or any other unproven hypothesis.

9 Performance Issues

9.1 One-time pad consumption

The use of one-time pads requires a significant amount of storage capacity. However, the required storage is quite reasonable, as shown by the following examples.

Two examples for OTPMAC parameters are shown in Table 1. We assume that both parties choose to be anonymous for all of these examples. For Example 1, we choose the OTPMAC parameter $p = 2^{31}-1$, which gives a security parameter $\sigma \approx 2^{-31}$. Using the identifier and maximum transaction/response sizes shown, 64 bytes of one-time pad are consumed in each transaction.

Table 1: Example parameter choices

Parameter	Example 1	Example 2
Security Parameter (σ)	$\approx 2^{-31}$	$\approx 2^{-127}$
OTPMAC prime (p)	2^{31}-1	2^{127}-1
One-time identifier size	32 bits	128 bits
SYNC size	32 bits	128 bits
Transaction size	22 bytes	28 bytes
Response size	13 bytes	14 bytes
One-time pad consumed per transaction	**64 bytes**	**128 bytes**

For Example 2, we choose the higher security value of $p = 2^{127}-1$, which gives a security parameter $\sigma \approx 2^{-127}$. The sizes of identifiers are also increased, as are the maximum transaction/response sizes. For these parameters, 128 bytes of one-time pad are used per transaction. These two examples are used to generate performance estimates for the various parties in the following sections.

9.2 Customer/CCC Performance

A customer using a smart card with 32K of memory can perform 512 transactions before its one-time pad is exhausted, under the parameter choices of Example 1. Even for a heavy card user (average 8 transactions per day), this card would last about two months before its one-time pad was exhausted.

Table 2: Performance for a user with a 32K smart card

Parameter	Example 1	Example 2
Number of transactions	512	256
OTP lifetime (8 per day)	≈ 2 months	≈ 1 month
CCC storage (1 million users)	32 GB	32 GB

The CCC also needs to store copies of all the one-time pads. Using these parameters, the CCC could service 1 million card users using only 32 GB of storage. This amount of storage is certainly reasonable, especially for a large transaction-processing facility. Table 2 details these estimates, and provides the adjusted estimates for the higher security of Example 2.

9.3 Merchant/CCC Performance

The number of transactions that can be performed using a smart card is probably insufficient for merchants. One alternative would be to distribute the one-time pad to the merchant on CD-ROM or other large media.

Using the parameters from Example 1 above, a merchant equipped with a CD-ROM one-time pad could perform 10.4 million transactions. Even performing 20,000 transactions per day, this pad would last 1.4 years. These examples are detailed in Table 3.

Table 3: Performance for a merchant with a CD-ROM one-time pad

Parameter	Example 1	Example 2
Number of transactions	10.4 million	5.2 million
OTP lifetime (20,000 per day)	≈ 1.4 years	≈ 8 months
CCC *offline* storage (5000 merchants)	3 TB	3 TB
CCC *online* storage (1 month)	180 GB	360 GB

Although the CCC needs to store copies of all the one-time pads, it is important to note that each one-time pad is accessed strictly sequentially. This means that the CCC might store only the active portion of each one-time pad in online storage, and the remainder offline. The last row of Table 3 estimates the online storage required to store sufficient one-time pad to process transactions for at least one month. This amount of storage is reasonable for a large processing facility. There are also many other possibilities for storage optimization.

9.4 CCC Identity Lookup

Upon receipt of M2, the CCC determines the identities of the merchant and customer using their one-time identities. One way for the CCC to perform this lookup is to use a hash table. The hash table lookup returns the real identities of the participants, which are used to retrieve the one-time pads. Once the transaction has completed, the hash table is updated with the (precomputed) *next* one-time identifiers for both parties.

Since the hash table contains only on a single one-time identifier for each user, it is much smaller than the entire collection of one-time pads. The size of the hash table is only a function of the identifier size, and not on the length of the one-time pads stored for each user. For the 32-bit identifiers and 100,000 users in the examples above, the hash table could occupy only about three megabytes, depending upon the hash implementation.

In the extremely unlikely event that multiple users correspond to the same one-time identifier, the CCC can perform the MAC verification for each of the candidate users. With overwhelming probability, the MAC will match for only one of the clients.

10 Related Work

Several other commerce and electronic cash protocols have been proposed. However, none of these provide the provable security and anonymity/unlinkability possible with one-time identifiers and OTPMAC.

Mondex International Ltd. has created the Mondex electronic cash system, which uses smart cards or other tokens with stored value[9]. Although the full protocol is undisclosed, there is a unique 16-digit identifier for each card, as well as an audit trail. These could reveal users' identities, or link a customer to multiple purchases.

DigiCash's eCash[11] is an electronic cash system that does provide anonymity and unlinkability of multiple transactions to a single party, even by the CCC. However, eCash requires RSA, which does not provide the provable security nor speed of OTPMAC.

The NetBill[13] commerce system is optimized for services delivered over a network, instead of face-to-face transactions. The NetBill transaction protocol uses eight messages, three more than required by our OTPMAC-based protocol. NetBill uses a static customer identifier, and provides neither provable security nor unlinkability.

The Millicent System[4][7] provides a commerce protocol optimized for small transactions over the Internet. Millicent uses a version of electronic cash called *scrip*, which is restricted to a specific vendor. Millicent requires at least three or four cryptographic operations per transaction; our protocol requires only two. Millicent provides neither provable security nor unlinkability.

SET[8][12] is a set of standards designed for secure credit card transactions. Since SET uses digital certificate chains, it requires many more signature verifications than our commerce protocol. Although SET can provide non-repudiation, SET's digital certificates provide neither provable security nor unlinkability.

References

[1] M. Bellare, R. Canetti, and H. Krawczyk, "Keyed Hash Functions and Message Authentication", *Advances in Cryptography, Crypto96 Proceeding*, June 1996. pp. 1-15. (http://www.research.ibm.com/security/keyed-md5.html)

[2] B. Cox, J.D. Tygar, M. Sirbu. "NetBill Security and Transaction Protocol." *Proceedings of the First USENIX Workshop on Electronic Commerce*

[3] D. Eastlake, 3rd, S. Crocker, J. Schiller. *RFC 1750: Randomness Recommendations for Security.* December 1994.

[4] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, P. Sobalvarro. "The MilliCent Protocol for Inexpensive Electronic Commerce." *Proceedings of the 4th International World Wide Web Conference.* December 1995.

[5] H. Krawczyk, M. Bellare, R. Canetti. *RFC 2104: HMAC: Keyed-Hashing for Message Authentication.* February 1997.

[6] C. Madson, R. Glenn. *RFC 2404: The Use of HMAC-SHA-1-96 within ESP and AH.* November 1998.

[7] M. S. Manasse. "The MilliCent Protocols for Electronic Commerce." *Proceedings of the 1st USENIX workshop on Electronic Commerce.* July 1995.

[8] M. S. Merkow, J. Breithaupt, K. Wheeler. *Building SET Applications for Secure Transactions.* John Wiley & Sons, New York. 1998.

[9] Mondex Electronic Cash. "How Private is a Mondex Transaction?" *Frequently Asked Questions* (via http://www.mondex.com)

[10] NIST, *FIPS PUB 180-1: Secure Hash Standard.* April 1995. (http://csrc.nist.gov/fips/fip180-1.txt)

[11] B. Schoenmakers. "Basic Security of the ecash Payment System." *Computer Security and Industrial Cryptography: State of the Art and Evolution.* ESAT Course, Leuven, Belgium. June 1997.

[12] SET Secure Electronic Transaction LLC. *SET Secure Electronic Transaction Specification.* December 1997.

[13] M. Sirbu, J. D. Tygar. "NetBill: An Internet Commerce System Optimized for Network Delivered Services." Prepared for the *IEEE CompCon Conference.* March 1995.