VOLUME 2, NUMBER 1 — SPRING 1996

RSA LABORATORIES'

CryptoBytes

The technical newsletter of RSA Laboratories, a division of RSA Data Security, Inc.

Contents

Asymmetric Encryption: Evolution and Enhancements

Don B. Johnson and Stephen M. Matyas
IBM Cryptography Center of Competence, MS P330
522 South Road
Poughkeepsie, NY 12601 USA

When public key cryptography was invented, one of its uses was identified as the secure transport of secret symmetric keys. The objectives of such a key transport mechanism keep evolving as attacks are identified, hidden assumptions are revealed, proofs of security are given, and additional capability is needed. The process continues in this article.

We trace the evolution of some asymmetric key transport mechanisms, starting with the method in PKCS #1 [10]. We then discuss, in historical order, two masking techniques developed by IBM cryptographers, and the method currently under study in ANSI draft standard X9.44 RSA Key Transport. We then give ideas that may be useful when using elliptic curve cryptography, where the size of the block is typically much less than that used with other algorithms, for example, RSA.

We will use the following terminology:

Formatted block – a block of data passed as input to the methods. It contains a secret symmetric key and other data to provide evidence of correct recovery and to thwart certain attacks.
Masked block – a block that results when the formatted block is masked to hide patterns.
Encrypted block – the block that results after the formatted or masked block has been asymmetrically encrypted.

For readers interested in some of the security issues involved in using RSA, an earlier CryptoBytes article entitled The Secure Use of RSA [9] contains much useful information.

PKCS #1

The Public Key Cryptography Standard #1 was designed by the cryptographers at RSA Data Security, Inc. [10]. PKCS #1 describes a method to RSA encrypt a secret symmetric key. The formatted block is passed directly to the RSA encrypt process. It uses the following method (with rationale):

1. A leading 0x00 is in the block to be RSA encrypted, ensuring the encryption block is less than the RSA modulus.
2. A block type encoded octet of 0x02 follows the leading 0x00, indicating the block is to be encrypted using a public key.
3. At least eight non-zero pseudorandom padding octets (bytes) are appended to the right after the block type octet. The padding octets should be generated independently for each RSA encryption, especially if the same key is being encrypted. This thwarts Hastad's attack [6] and allows use of a low value (e.g., 3) for the public

(continued on page 3)

Don Johnson, a senior programmer at IBM, is an architect of IBM's crypto solutions and the company's representative to ANSI X9.F.1 and X9.F.3, IEEE P1363, and the X/Open Crypto API workgroups. He can be contacted at dbj@VNET.IBM.COM. Mike Matyas, a senior technical staff member at IBM, has participated in the design and development of all major IBM cryptographic products. He can be contacted at smatyas@VNET.IBM.COM.

Editor’s Note

Readers of the Autumn 1995 issue of CryptoBytes will recall that we included two articles related to RSA encryption. In one, we concentrated on the secure use of conventional RSA and in the other we looked at a new proposal by Adi Shamir termed Unbalanced RSA.

We encourage any readers with comments, opposite opinions, suggestions or proposals for future issues to contact the CryptoBytes editor.

Our lead article in this issue is also related to RSA encryption, but in reality has wider implications for public-key technology in general. The use of what was introduced as Optimal Asymmetric Encryption Padding (OAEP) has been receiving a lot of attention recently and it seems an appropriate time to look at this topic in more depth. We lead this issue with an article by Don Johnson and Stephen Matyas on the evolution of padding schemes for use with asymmetric encryption. Many readers will already be aware that standardization efforts are currently underway on the specification of such techniques (for instance as part of the current Secure Electronic Transactions (SET) standard effort) and we hope that this article will provide readers with some of the background to these discussions. Future developments will, of course, be reported in upcoming issues of CryptoBytes.

For new technologies we turn our attention to what have been called micropayment schemes and to two that were recently devised by Ron Rivest and Adi Shamir. Micropayment techniques are aimed specifically at accommodating commerce over the Internet and World-Wide Web where small-scale payments are often required at an exceptionally low operational cost. In this article, the significant features of PayWord and MicroMint are described as well as the philosophy behind some of the security issues involved.

Our third article is a presentation of HMAC by Mihir Bellare, Ran Canetti and Hugo Krawczyk. In the use of keyed-MD5 for message authentication, HMAC appears to be a solution that is rapidly gaining acceptance and a report on this construction, particularly since it forms a continuation of previous CryptoBytes articles, will be of interest to a great many readers.

This issue of CryptoBytes is the first of what will become the second volume. In the year since the newsletter was launched we have had a lot of positive response to both the aims of the newsletter and to the articles we have been carrying. We are delighted to announce that CryptoBytes will now be available free of charge and that all issues will be directly accessible via the World-Wide Web.

The future success of CryptoBytes depends on input fom all sectors of the cryptographic community, and as usual we would very much like to thank the writers who have contributed to this first issue of the second volume. We encourage any readers with comments, opposite opinions, suggestions or proposals for future issues to contact the CryptoBytes editor at RSA Laboratories or by E-mail to bytes-ed@rsa.com.

Newsletter Availability and Contact Information

CryptoBytes is a free publication and all issues, both current and past, are available via the World-Wide Web at http://www.rsa.com/rsalabs/cryptobytes/.

For each issue a limited number of copies are printed. They are distributed at major conferences and through direct mailing. While available, additional copies of the newsletter can be requested by contacting RSA Laboratories though a nominal fee to cover handling costs might be charged for individual requests.

RSA Laboratories can be contacted at:

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com

About RSA Laboratories

RSA Laboratories is the research and development division of RSA Data Security, Inc., the company founded by the inventors of the RSA public-key cryptosystem. RSA Laboratories reviews, designs and implements secure and efficient cryptosystems of all kinds. Its clients include government agencies, telecommunications companies, computer manufacturers, software developers, cable TV broadcasters, interactive video manufacturers, and satellite broadcast companies, among others.

Asymmetric Encryption
Continued from page 1

key exponent, which is important for performance reasons.

4. An octet of 0x00 is then appended on the right of the above to act as a delimiter between the non-zero pseudorandom octets and the key.
5. The secret symmetric key is then appended on the right of the above.

A simplified diagram of the PKCS #1 method follows:

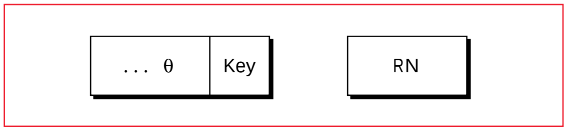

Figure 1. Simplified PKCS #1 method. Key is the secret symmetric key and RN is a random number. The actual specification of PKCS #1 differs from this simplified representation. See the text for details.

After RSA decryption, an error is returned if any of the following conditions exist:

1. The block type octet is not 0x02.
2. There are not at least eight padding octets. A padding octet cannot be all binary zeros.
3. The block cannot be parsed. Besides the above reasons, this might happen if a zero octet delimiter is not found.

There is significant security provided by the PKCS #1 method. We are unaware of any realistic attacks on this method in practise. Placing the non-zero block type octet near the left end of the block ensures that the RSA modulo reduction will occur, even for low public key exponents, like 3. There are at least 64 bits of pseudorandomness to thwart an exhaustion attack and Hastad's attack. There are a minimum of 16 bits of redundancy (block type octet and zero delimiter octet) to provide evidence that the formatted block has been recovered correctly.

However, a concern could be raised that some of the bits in the formatted block are known. Recent results by Don Coppersmith [4] show that for an RSA public exponent of 3 and an n-bit modulus, if an attacker knows all but $^{n}/_{3}$ of the plaintext and all the ciphertext, he can recover all the plaintext; if an attacker knows that two messages agree in all but $^{n}/_{9}$ bits and are encrypted with the same public key, he can recover the plaintext.

Readers interested in the development of PKCS#1 are referred to the editors note at the conclusion of this article.

IBM Transaction Security System

At about the same time as the PKCS was being published, IBM developed a public key extension to their Transaction Security System (TSS) to support digital signatures and key transport. Cryptographers at IBM (Johnson, Le, Martin, Matyas, and Wilkins) developed a method to mask a formatted block before its being RSA encrypted [7, 8]. A simplified diagram of the JLMMW method is given in Figure 2.

Its objectives, as given in the paper, are as follows:

1. Ensure the method is strong for any arbitrary public key value, large or small.
2. Ensure that the method is strong regardless of whether the same secret symmetric key value is encrypted by many different public keys.
3. Ensure there are least at 32 bits of redundant data that may be used to authenticate the symmetric key recovery process.
4. Remove any structure from the data that is encrypted by the public key.
5. Ensure the symmetric key can always be encrypted as one block by the public key.

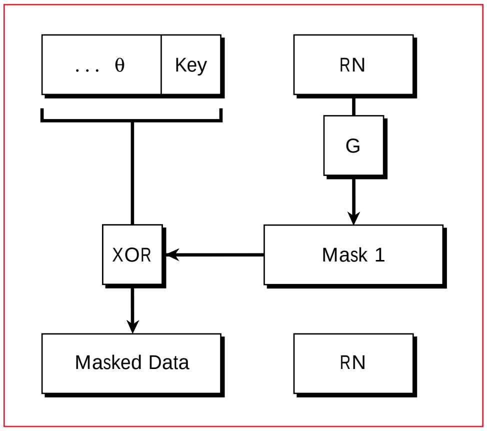

Figure 2. Simplified JLMMW method. RN is a random number and G is a masking function that produces a mask from a supplied random input.

Goal 1 is a common goal of most asymmetric systems, as often some keys will have better performance than others. Goals 2, 3, and 5 are also goals of the PKCS #1 design. Goal 3 was addressed in the PKCS #1 method by using 16 bits and in the JLMMW method by using 32 bits.

Goal 4 was not addressed in PKCS #1 and is the most unusual. In fact, goals 3 and 4 appear to conflict. Goal 3 says there is redundancy (structure) to check and goal 4 says there does not appear to be any structure. The solution is to realize that while there may need to be significant structure, there should not APPEAR to be any structure. This implies a masking operation is performed on the structured data so that it appears

At about the same time as the PKCS was being published, IBM developed a public key extension to their Transaction Security System to support digital signatures and key transport.

While there is significant structure in the formatted block, the masked block appears random.

random. The idea is to provide a random number in the formatted block, that would be input to a pseudorandom generator and used to generate a pseudorandom string that could be exclusive-ORed with the formatted block, thereby masking any structure. A different random number is generated each time a formatted block is to be RSA encrypted. (This ensures that a formatted block is not encrypted twice using different RSA public keys.) In effect, RSA encryption is performed on masked data consisting of a random appearing string concatenated to a random number. While there is significant structure in the formatted block, the masked block appears random.

Besides the basic masking method, specifying control information was identified as an option.

Optimal Asymmetric Encryption

Other cryptographers at IBM (Bellare and Rogaway) saw a way to improve the JLMMW construction [3]. Fig. 3 gives a simplified diagram of their construction.

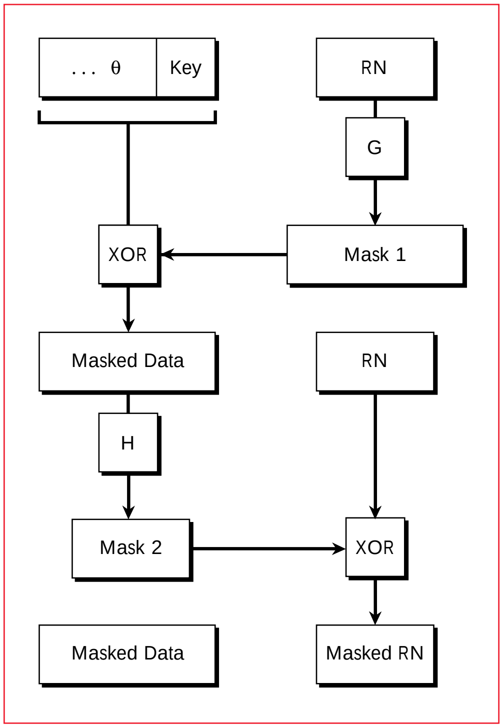

Figure 3. Simplified Bellare-Rogaway OAE method. G and H are masking functions.

The question they asked themselves was: Suppose an adversary could recover some of the bits in the encrypted block more easily than recovering the entire encrypted block. If this were possible, then this knowledge might enable one to more easily deduce some or all of the remaining bits: that is, it might give an advantage in recovering other bits.

A result given by Alexi, Chor, Goldreich and Schnorr suggests that finding the low order bits of an RSA encrypted block are as hard as finding the entire encrypted block [1]. But what if an adversary attacked bits not covered by this result, or what if another asymmetric algorithm was used, one possibly without this property?

Although they did not find an attack, they apparently had the following concern about the JLMMW masking method: If an adversary could discover the random number component of the masked block, then an adversary could reconstruct the masking string. Knowing the masking string and the structure in the formatted block, it might be possible for an attacker to deduce other parts of the formatted block. Their insight into the construction of a good masking function was to realize that one could also protect the random number by masking it with a hash value generated on the masked block (less the random number). If this was done in the right way, then an adversary would have to recover the entire masked block in order to invert the masking operation and recover the formatted block containing the secret symmetric key. They were then able to provide a proof of the security of their method.[1]

To provide for non-malleability, 128 binary zeroes are used in the formatted record.[2] This also provides evidence of correct recovery.

Enhanced Optimal Asymmetric Encryption

The enhancement over the OAE method provided in the proposed ANSI X9.44 draft standard is essentially to add a one-way hash of information associated with the key, such as higher-level security protocol data [2]. As this also provides for non-malleability, the 128 bits of binary zeros in the formatted block, called for by Bellare and Rogaway, may be omitted.

While still providing non-malleability, this provides a strong coupling mechanism of the secret symmetric key to arbitrary information associated with the key. For example, such information may include:

1. Method version identification.
2. Identification of the H and G masking functions.
3. Type of symmetric key algorithm.
4. Intended usage of the symmetric key.
5. Cryptoperiod for the key.
6. Identifiers for the creator and/or recipient of the key.
7. Key creation timestamp.

[1] DES may be viewed as a ladder with 16 rungs, each rung corresponding to a round of DES. This is also called a Feistel network or ladder. On the odd numbered rounds, a one-way function (under the control of the key) operates on the rightmost 32 bits to produce a mask that is exclusive-ORed with the leftmost 32 bits. On the even numbered rounds the one-way function operates on the (masked) leftmost 32 bits to produce a mask that is exclusive-

8. A nonce used in a request/response protocol to ensure freshness.

As there are many possible ways to use this capability, it is important that it be open-ended. As new protocols are devised and the need to strongly couple currently unforeseen information to the secret key becomes identified, this method can grow to meet potential future requirements. A simplified diagram of this mechanism is shown in Figure 4.

Reverse Signature Concept

One way to perceive the enhancement of coupling a hash of information associated with a key to the key itself is as a kind of "reverse signature." In a normal digital signature, only the owner of the private key can generate a signature, but anyone can have access to the public key to verify the signature. In a "reverse signature," anyone can generate such a signature, but only the owner of the private key can verify it.

Using asymmetric key cryptography, there are two ways to cryptographically couple information to a key: by using a digital signature and by using a reverse signature.

However, there are several distinct advantages to the reverse signature concept:

1. The implementation of key recovery is simplified, needing only the associated private key. It does not need a public key to verify the recovery or to provide coupling to associated information about the key.

2. All digital signatures may be calculated on the encrypted symmetric key block. None need to be calculated on the unencrypted symmetric key block. This ensures all parties can verify any digital signatures as needed, without requiring recovery of the symmetric key. This can be important in a generalized environment with many certification authorities in varying relationships.

3. In an interactive environment, one may be able to avoid the use of digital signatures. In this scenario, the initiator RSA-encrypts a secret nonce using the public key of the secret key generator. The secret key generator is the only entity that can recover the secret nonce, so only the initiator and the secret key generator know it. The secret key generator generates the secret key and includes the secret nonce in the RSA encrypted reply. The initiator then supplies the secret nonce to the secret key recovery process. The initiator is assured that the secret key is from the key generator. The key generator is assured that only the initiator is able to recover the secret key.

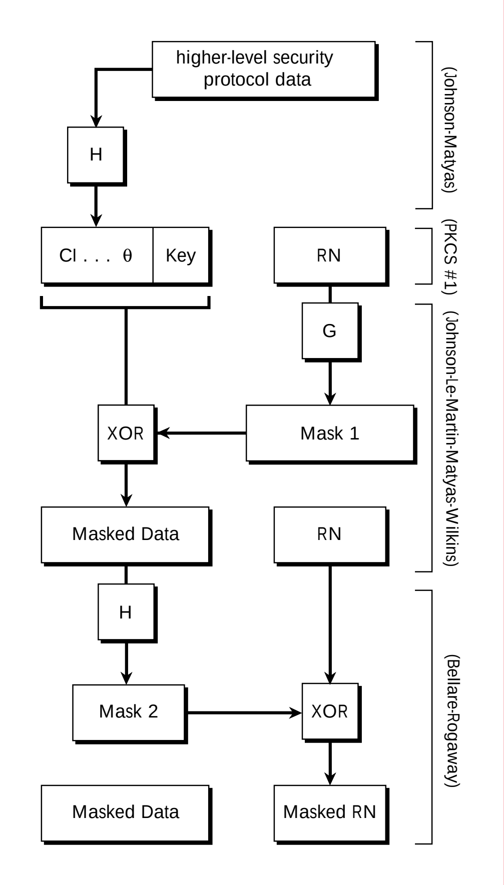

Figure 4. Simplified Johnson-Matyas X9.44 draft method

4. As the information associated with the key is supplied to the key recovery process, it is expected to be easy to demonstrate that this method provides secrecy only for a symmetric key of limited size. This can be important due to product export considerations.

ORed with the rightmost 32 bits. Another way to perceive the improvement that the OAE method provided over the JLMMW method is to realize that the JLMMW method is analogous to a single round of DES encryption, while the OAE masking method is analogous to two rounds of DES encryption. That is, with the JLMMW method, the left part (A) was masked using the right part (B); with the BR method, the left part (A) was masked using the right part (B) and the right part (B) was masked using the masked left part (masked A).

[2] Non-malleability was defined in a paper by Dolev, Dwork, and Naor [5]. Informally, a cryptosystem is non-malleable if it is no easier to determine some plaintext A in some relation to some unknown plaintext B if one knows the ciphertext of B than if one does not. An example given in the paper uses two bidders encrypting their bids. It should not be possible for one bidder to see the encrypted bid of the other and somehow be able to offer a bid that was slightly lower, even if he would not know what the resulting bid actually was.

Further Enhancements

The recent emergence of practical cryptosytems based on elliptic curves has resulted in other potential refinements. These are mainly because elliptic curve cryptographic algorithms using a blocksize of about 155 bits are thought to be strong; however, this means that the abundance of space in a block that allows the ability to store lots of data like a symmetric key, a reverse signature, and a random number when using larger blocksizes is severely limited. The following are some ideas to help address this and are under consideration by the IEEE P1363 workgroup:

1. The Feistel ladder can be balanced, rather than unbalanced. That is, the dividing line between the two sides for the masking process operating on the unformatted block could be put right in the middle. By symmetry considerations, a balanced construct is preferable from a security viewpoint as it ensures optimal distribution of the random secret data throughout the block. There is no "funnel" in which secrecy may be reduced. It may also be easier to implement. Also, by freeing the dividing line between the two parts from the possibly arbitrary lengths of certain fields, a reduction in overall length may be able to be achieved.
2. The random number is used for 2 purposes, to ensure uniqueness and to help thwart dictionary attacks. However, use of a random number has a concern in that the uniqueness is probabilistic, not guaranteed. Also the random number must be long enough so that the chance of a duplicate is sufficiently small. If the symmetric key being sent is already large enough to thwart dictionary attacks, then the random number may be able to be replaced with a counter or some other way to ensure uniqueness of the block. Not only will this result in a smaller size than the use of a random number, uniqueness can be guaranteed.
3. One may consider doing more than two rounds of masking to achieve some desired property.

Conclusion

In this article we have presented the increases in security and capability that have occurred when using asymmetric encryption. The advantageous attributes of these ideas are such that we believe they will be able to be used in a wide variety of cryptographic solutions.

Acknowledgments

We thank Mihir Bellare for helping us with the ANSI X9.44 proposal to ensure it incorporated ideas in "Optimal Asymmetric Encryption." We thank Phillip Rogaway for confirming that using a hash function on public information associated with the secret key supports security enhancements over the basic OAE method.

References

[1] W. Alexi, B.Z. Chor, O. Goldreich, and C.P. Schnorr, RSA and Rabin Functions: Certain Parts are as Hard as the Whole, SIAM Journal on Computing, 17(2), 194-209, April 1988.

[2] Draft ANSI X9.44 RSA Key Transport standard, October 1994.

[3] M. Bellare and P. Rogaway, Optimal Asymmetric Encryption – How to Encrypt with RSA, in A. De Santis, ed., Advances in Cryptology – Eurocrypt '94 Proceedings, volume 950 of Lecture Notes in Computer Science, Springer-Verlag, New York, 1994.

[4] D. Coppersmith, Finding a Small Root of a Univariate Modular Equation, IBM Research Report RC 20223, October 11, 1995; revised November 8, 1995.

[5] D. Dolev, C. Dwork, M. Naor, Non-Malleable Cryptography, Proceedings of the 23rd ACM Symposium on Theory of Computing, 1991.

[6] J. Hastad, Solving simultaneous modular equations, SIAM Journal on Computing, 17(2), 336-341, April 1988.

[7] Common Cryptographic Architecture: Cryptographic Application Programming Interface – Public Key Algorithm, IBM publication SC40-1676, April 1993.

[8] D. Johnson, A. Le, W. Martin, S. Matyas, and J. Wilkins, Hybrid key distribution scheme giving key record recovery, IBM Technical Disclosure Bulletin, 37(2A), 5-16, February 1994.

[9] B.S. Kaliski and M.J.B. Robshaw. The Secure Use of RSA. RSA Laboratories' CryptoBytes, 1:3, 7-13, Autumn 1995.

[10] RSA Laboratories, Public Key Cryptography Standard #1: RSA Encryption Standard Version 1.5, November 1993.

Editor's Note: We consider this article by Johnson and Matyas to be particularly timely. Not only are a variety of efforts already underway for standardizing on a version of the optimal asymmetric encryption methods, but RSA Laboratories is also currently preparing to revise PKCS #1 – our public-key cryptography standard for RSA encryption and signatures. Progress on all these efforts will be reported in forthcoming issues of CryptoBytes and in other RSA Laboratories' publications.

PayWord and MicroMint *(extended abstract)*

Ronald L. Rivest
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139 USA

Adi Shamir
Applied Math Department
The Weizmann Institute of Science
Rehovot 76100, Israel

Many electronic payment schemes have been proposed in recent years to address the problem of secure transactions over open networks. Micropayment schemes are a special kind of payment schemes for applications in which each payment is very small. To support micropayments, exceptional efficiency is required; otherwise, the cost of the mechanism will exceed the value of the payments.

PayWord and MicroMint are two simple micropayment schemes. In both schemes, the players are brokers, users and vendors. Brokers authorize users to make micropayments to vendors and redeem the payments collected by the vendors (See Figure 1). Broker-user and broker-vendor relationships are long-term, while user-vendor relationships are transient.

Our micropayment schemes might be considered lightweight when compared with full payment schemes in the sense that our security goal is to keep honest people honest (a similar situation exists with newspaper vending machines): exceptionally small-scale fraud and abuse cannot be totally prevented. In return, however, we try to achieve the following efficiency goals:

- Minimize the number of public-key operations by using hash operations whenever possible. (Roughly, hash functions are about 100 times faster than RSA signature verification and about 10,000 faster than RSA signature generation.)
- Minimize communication (particularly on-line communication) with brokers. Since there may be only a few nationwide brokers, it is important that their computational burden be both reasonable and "off-line."
- Make the communication per transaction efficient, especially for repeated small purchases.

PayWord is a credit-based scheme and is optimized for sequences of micropayments. It is also secure and flexible enough to support larger variable-value payments as well. MicroMint introduces a new paradigm for electronic coins and is designed to eliminate public-key operations altogether. It has lower security but higher efficiency.

...our security goal is to keep honest people honest...

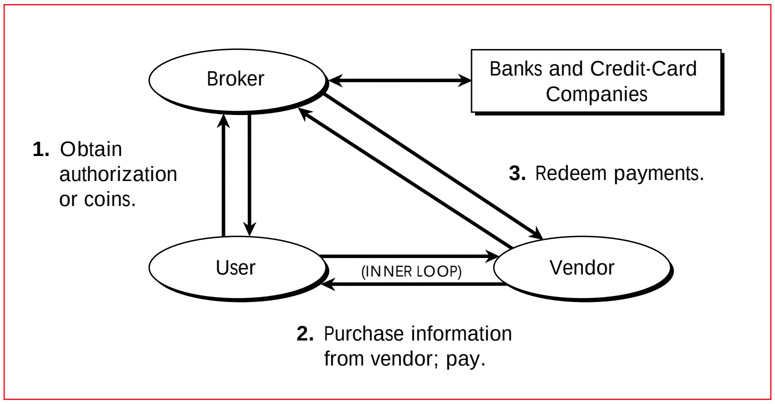
Broker
Banks and Credit-Card Companies
1. Obtain authorization or coins.
3. Redeem payments.
User
(INNER LOOP)
Vendor
2. Purchase information from vendor; pay.

Figure 1. Broker/Vendor/User Relationship

PayWord

Roughly speaking, PayWord consists of the following components:

1. A user establishes an account with a broker, who issues the user a digitally-signed certificate. This authorizes the user to make payword chains, which are chains of hash values.
2. Before contacting a vendor, the user creates a vendor-specific payword chain off-line.
3. The user authenticates the complete chain to the vendor with a single public-key signature, and then successively reveals each payword in the chain to make each micropayment.
4. The vendor redeems the paywords received from the user with the original broker.

Payword Chains

In the construction of a payword chain, we will use a one-way hash function, such as MD5 [6]. The hash function h has the property that, given a hash value y it is hard to find an input x, such that y=h(x).

Ron Rivest is associate director of MIT's Laboratory for Computer Science. He can be contacted at rivest@theory.lcs.mit.edu. Adi Shamir is professor at the Applied Math Department of the Weizmann Institute of Science, Israel, and can be contacted at shamir@wisdom.weizmann.ac.il. Rivest and Shamir are co-inventor of the RSA cryptosystem.
This article is edited from the full paper [7] by Yiqun Lisa Yin. She is a research scientist at RSA Laboratories, and can be reached at yiqun@rsa.com.

A user creates a payword chain $\{w_0, w_1, w_2, \ldots, w_n\}$ by picking the final payword w_n at random and computing paywords according to $w_i = h(w_{i+1})$. The first payword w_0 is called the "root" of the chain. Given w_0 it is hard for anyone other than the user to figure out the rest of the paywords in the chain. By revealing w_0, however, the user is "committed" to the payword chain she just created.

User-Broker relationship

A user begins a relationship with a broker by requesting an account and a PayWord Certificate. The user first gives the broker over a secure authenticated channel: her credit-card number, her public key PK_U, and her "delivery address" (e.g., IP-address). The broker then issues the user a digitally signed certificate which authorizes her to make payword chains until a given expiration date, and authorizes the delivery of goods only to the specified delivery address.

The user's certificate C_U thus has the following form:

$$C_U = \{\text{broker, user, user's delivery address, } PK_U\text{, expiration-date, other-info}\}_{SK_B},$$

User-vendor relationships are transient. A user might visit a web site, purchase ten pages, and then move on elsewhere.

where $\{\,\}_{SK_B}$ denotes that the contents of $\{\,\}$ are signed with the broker's private key SK_B. The PayWord certificate C_U is a statement by the broker to any vendor that authentic paywords produced by the user and used before the expiration date will be redeemed.

User-Vendor relationship

User-vendor relationships are transient. A user might visit a web site, purchase ten pages, and then move on elsewhere. When the user is about to contact a new vendor, she computes a fresh payword chain $\{w_0, w_1, w_2, \ldots, w_n\}$ Here n is chosen at the user's convenience; it could be ten or ten thousand. She then computes her commitment for that chain:

$$M = \{\text{vendor}, C_U, w_0, \text{current-date, other-info}\}_{SK_U}.$$

The commitment M, which is signed with the user's private key SK_U, authorizes the broker to pay the vendor for any of the paywords $w_1, w_2, \ldots, w_n$ redeemed on the current date. Paywords are vendor-specific and user-specific; they are of no value to another vendor. Upon receiving the commitment M, the vendor verifies the user's signature on M and the broker's signature on C_U (contained within M), and checks the expiration dates. After the commitment, payment from a user to a vendor consists of a payword and its index: (w_i, i). The payment is not signed by the user.

The user spends her paywords in order: w_1 first, then w_2, and so on. If each payword is worth one cent and each web page costs one cent, then she discloses w_i to the vendor when she orders her i-th web page from the vendor that day. This leads to the PayWord payment policy: For each commitment a vendor is paid l cents, where (w_l, l) is the corresponding payment received with the largest index. This means that the vendor needs to store only one payment from each user: the one with the highest index. The broker can confirm the value to be paid for w_l by determining how many applications of h are required to map w_l into w_0.

Vendor-Broker relationship

A vendor needn't have a prior relationship with a broker, but does need to obtain the public key of the broker in an authenticated manner, so he can authenticate certificates signed by the broker. He also needs to establish a way for the broker to pay him redeemed paywords.

At the end of each day (or other suitable period), the vendor sends the broker a redemption message giving, for each of the broker's users who have paid the vendor that day, the commitment C_U and the last payment $P = (w_l, l)$. The broker needs to first verify each commitment received by checking the user's signatures (since he can recognize his own certificates), and then verify each payment (w_l, l) by computing the hash function l times. The broker will normally honor all valid redemption requests.

Security

In a typical scenario for PayWord, each payword represents a very small amount of money, such as one cent. After the commitment, the paywords do not need to be signed by the user, since the paywords are self-authenticating using the commitment. Such a feature may allow each payword to represent a larger amount of money (e.g., software selling at $19.99). Hence, the level of security and flexibility of PayWord makes it possible to support certain "macro-payment" as well.

In PayWord, the contents of a payment does not specify what item it is payment for. A vendor may cheat a user by sending her nothing, or the wrong item, in return. The user bears the risk of losing the payment, just as if she had put a penny in the mail. Vendors who so cheat their customers will be shunned. This risk can be moved to the vendor, if he specifies payment after the document has been delivered. If the user doesn't pay, the vendor can notify the broker and/or refuse the user further service. For micropayments, users and vendors might find either approach workable.

Efficiency

PayWord is designed to minimize the on-line communication with brokers: the vendor does not need to interact with the broker when a user first contacts the vendor, nor is any interaction with the broker required as each payment is made. In PayWord, the communication per transition is also very efficient: after the commitment, each payword is only 20 to 30 bytes long. Moreover, the broker does not even receive every payword spent, but only the last payword spent by each user each day at each vendor. PayWord is thus extremely efficient when a user makes repeated requests from the same vendor, but is quite effective in any case.

PayWord is thus extremely efficient when a user makes repeated requests from the same vendor, but is quite effective in any case.

PayWord's computational and storage requirements are summarized below:

- The broker needs to sign each user certificate, verify each user commitment, and perform one hash function application per payment. (All these computations are off-line.) The broker stores copies of user certificates and maintains accounts for users and vendors.

- The user needs to verify her certificates, sign each of her commitments, and perform one hash function application per payword committed to. (Only signing commitments is an on-line computation.) She needs to store her private key, her active commitments, the corresponding payword chains, and her current position in each chain.

- The vendor verifies all certificates and commitments received, and performs one hash function application per payword received or skipped over. (All these computations are on-line.) The vendor needs to store all commitments and the last payment received per commitment each day.

MicroMint

MicroMint introduces a new paradigm for representing electronic coins by "hash function collisions." In fact, it uses hash functions only without public-key cryptography. MicroMint has the property that generating many coins is much cheaper, per coin generated, than generating few coins. A large initial investment is required to generate the first coin, but then generating additional coins can be made progressively cheaper. This is similar to the economics for a regular mint, which invests in a lot of expensive machinery to make coins economically.

MicroMint introduces a new paradigm for representing electronic coins by "hash function collisions."

In a typical setting, a broker produces coins that are valid for a given period (e.g., a month). The broker needs to invest in substantial hardware that gives him a computational advantage over would-be forgers, and run this hardware continuously for a month to compute coins valid for the next month. New coins are issued to users at the beginning of each month. Users give valid coins to vendors as payment, and vendors return coins collected to the broker at their convenience (e.g., at the end of each day).

MicroMint coins as hash function collisions

MicroMint coins are represented by hash function collisions, for some specified one-way hash function h mapping m-bit strings x to n-bit strings y. A k-way collision (k • 2) is a set of k distinct x-values ($x_1, x_2, \ldots, x_k$) that have the same hash value y.

We first give a brief review of the computation time for finding hash function collision. Assume h acts "randomly," then the only way to produce even a single k-way collision is to hash about $2^{n(k-1)/k}$ x-values and search for repeated outputs. (The case for 2-way collisions is essentially the "birthday paradox.") However, if one examines c times this many x-values, for $1 < c < 2^{n/k}$, one expects to see about c^k k-way collisions. Hence, producing collisions can be done increasingly efficiently, per coin generated, once the first collision has been passed. Note that increasing k has the dual effect of delaying the threshold at which the first collision is seen, and also accelerating the rate of collision generation once the threshold is passed.

We thus let a k-way collision ($x_1, x_2, \ldots, x_k$) represent a coin. This coin can be easily verified by checking that the x_i's are distinct and that

$$h(x_1) = h(x_2) = \ldots = h(x_k) = y$$

for some n-bit string y.Note that each coin is a bit-string whose validity can be easily checked by anyone, but which is hard to produce. This is similar to the requirements for a public-key signature, but the complexity of the signature makes it an overkill for a transaction whose value is only one cent.

The process of minting coins

The process of computing $h(x)=y$ is analogous to tossing a ball x at random into one of 2^n bins; the bin which ball x ends up in is the one with index y.A coin is thus a set of k balls that have been tossed into the same bin. Getting k balls into the same bin requires tossing a substantial number of balls altogether, since balls can not be “aimed”. To mint coins, the broker will create 2^n bins, toss approximately $k2^n$ balls, and create one coin from each bin that now contains at least k balls. In this way each ball has a chance of roughly 1/2 of being part of a coin.

Note that each coin is a bit-string whose validity can be easily checked by anyone, but which is hard to produce.

A small problem in this basic picture, however, is that computation is much cheaper than storage. The number of balls that can be tossed into bins in a long computation far exceeds the number of balls that can be memorized on a reasonable number of hard disks. We thus propose to make most balls unusable as part of a coin, in a manner that depends on the hash value y.To do so, we say that a ball x is “good” if the high-order t bits of the hash value y have a value z specified by the broker. More precisely, let $n = t + u$. If the high-order t bits of y are equal to z, then y is called “good,” and the low-order u bits of y determine the index of the bin in which the (good) ball x is tossed.

A proper choice of t enables us to balance the computational and storage requirements of the broker. This slows down the generation process by a factor of 2^t, without slowing down the verification process. The broker thus tosses approximately $k2^n$ balls, memorizes about $k2^u$ good balls that he tosses into 2^u bins, and generates from them about $(1/2) \infty 2^u$ valid coins.

A typical scenario

Here is a sketch of how a typical broker might choose the parameters (k, u, t, n) and make investment in hardware. We suppose that the broker wishes to have a net profit of $1 million per month, and he charges a 10% brokerage fee to vendors for redemption. Thus, the broker needs to sell one billion coins (approximately 2^{30}) per month to collect his $1 million fee. (This requires a customer base of 500,000 if an average user buys 2,500 coins for $25 per month.)

The broker first chooses $k = 4$; a coin will be a good 4-way hash collision. He then chooses $u = 31$ (i.e., creates 2^{31} bins), since this will allow him to create about $(1/2) \infty 2^{31} = 2^{30}$ coins.

The broker chooses his hash function $h(x)$ as the encryption of some fixed value v with key x under DES, which can be implemented by so-called field-programmable gate array chips. Each chip costs about $400 and computes 2^{25} values per second. Buying 2^8 of these chips costs the broker $100,000 and allows him to compute about 2^{54} values per month. Since $k = 4$, the broker chooses $n = 52$ and $t = 21$. Storing all good pairs $(x, h(x))$ requires less than 2^{37} bytes, and costs less than $40,000 using standard magnetic hard disk technology. Hence, the total cost for the broker’s hardware is less than $150,000, which is less than 15% of the first month’s profit.

Selling coins, making payments, and redemption

At the beginning of each month, the broker either reveals a new hash function h or changes the value z for that month and sells coins to users. Such sales can be on a debit basis or a credit basis, since the broker can recognize coins when they are returned to him for redemption. In a typical purchase, a user might buy $25.00 worth of coins (2500 coins), and charge the purchase to his credit card. The broker keeps a record of which coins each user bought. Unused coins are returned to the broker at the end of each month.

Each time a user purchases a web page, he gives the vendor a previously unspent coin $(x_1, x_2, \ldots, x_k)$. The vendor verifies that it is indeed a good k-way collision by computing k hash values. The vendor returns the coins he has collected to the broker at the end of each day. If the coin has not been previously returned, the broker pays the vendor one cent (minus any brokerage fee) for each coin. If a coin is received more than once, the broker does not pay more than once. Which vendor gets paid can be decided arbitrarily or randomly by the broker.

Security analysis

We believe that users and vendors will have little motivation to cheat in order to gain only a few cents; even if they do, the consequences are of no great concern. Our security mechanisms are thus primarily designed to discourage large-scale attacks, such as massive forgery or persistent double-spending.

...users and vendors will have little motivation to cheat in order to gain only a few cents...

Forgery: Can an adversary forge MicroMint coins? (Economically?) First, the computational difficulty of minting coins makes small-scale forgery not really a concern. (For the parameter choice given above, a forger would need to spend 80 years on a standard workstation just to generate the first coin). Second, large-scale forgers can be detected and countered. In particular, coins "expire" monthly, and the new hash function for each month is revealed only at the beginning of that month. (The broker works during May to make coins good for June; forger only learns the hash function at the beginning of June and so starts out way behind.) Additional protection against forgery may be obtained by restricting coins to satisfy "hidden predicates" which are only announced if forgery is detected by the broker. For example, legitimate coins may all satisfy condition that the low-order bit of x_i is equal to some complicated function of other bits. Forger's coins will typically not pass this additional "verification condition".

Double-spending: What if a user "double-spends" his MicroMint coins? There is no "anonymity" in MicroMint: the broker keeps track of whom each coin was sold to and notes when it is returned by vendor. Small-scale double-spending is not a concern. A user whose coins are consistently double-spent will be caught and black-listed; he will not be sold any more MicroMint coins.

Vendor fraud: What if a vendor gives copies of coins received to an accomplice? Vendors who consistently redeem coins that are also redeemed by other vendors will be black-listed and refused further redemption service by the broker. Users might cooperate with the broker to identify bad vendors by identifying where coins were first spent.

Theft of coins. If theft of coins is judged to be a problem during initial distribution to users or during redemption by vendors, it is easy to transmit coins in an encrypted form. Since user/broker and vendor/broker relationships are relatively stable, long-term encryption keys (e.g., DES keys) can be arranged.

To prevent theft of coins as they are being transferred from user to vendor, we can make coins user-specific so that they are of no value to other users. To produce such coins, we generalize the notion of a "collision" to more complicated combinatorial structures which are related to the identity of a user and can be easily checked during verification. The user-specific coins can be further modified so that they are vendor-specific as well, making a stolen coin even less desirable. Details of these extensions are in the full version of the paper [7].

Conclusions

In this article we have presented two new micropayment schemes which are exceptionally economical in terms of the number of public-key operations employed. Furthermore, both schemes are off-line from the point of view of the broker. The area of micropayments has attracted considerable attention recently, and several researchers have proposed related schemes (Millicent [4], NetBill [2], NetCard [1], Pedersen's tick payments [5], and a micropayment scheme based on iKP [3], etc). More details about our schemes and the relationships between the various proposals are described and analyzed in the full version of this extended abstract [7].

Our security mechanisms are thus primarily designed to discourage large-scale attacks, such as massive forgery or persistent double-spending.

References

[1] R. Anderson, H. Manifavas, and C. Sutherland. NetCard – A practical electronic cash systems. 1996. Available from author: Ross.Anderson@cl.cam.ac.uk.

[2] B. Cox, J.D. Tygar, and M. Sirbu. NetBill security and transaction protocol. http://www.ini.cmu.edu/netbill/home.html.

[3] R. Hauser, M. Steiner, and M. Waidner. Micro-payments based on iKP. January 1996. http://www.zurich.ibm.com:80/Technology/Security/extern/ecommerce/iKP.html.

[4] M. Manasse. The Millicent protocols for electronic commerce. 1995. http://www.research.digital.com/SRC/millicent.

[5] T. Pedersen. Electronic payments of small amounts. Technical Report DAIMI PB-495, Aarhus University, Computer Science Department, August 1995.

[6] R. Rivest. RFC 1321: The MD5 Message Digest Algorithm. RSA Data Security, Inc., April 1992.

[7 R. Rivest and A. Shamir. PayWord and MicroMint– Two simple micropayment schemes. April 1996. http://theory.lcs.mit.edu/~rivest.

The HMAC Construction

Mihir Bellare
Department of Computer Science & Engineering
Mail Code 0114, University of California at San Diego
9500 Gilman Drive, La Jolla, CA 92093 USA

Ran Canetti
Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139 USA

Hugo Krawczyk
IBM T.J. Watson Research Center, PO Box 704
Yorktown Heights, New York 10598 USA

There has recently been a lot of interest in the subject of authenticating information using cryptographic hash functions like MD5 and SHA-1, particularly for Internet security protocols. We report on our HMAC construction [1] which seems to be gaining acceptance as a solution.

Introduction

Two parties communicating across an insecure channel need a method by which any attempt to modify the information sent by one to the other, or fake its origin, is detected. Most commonly such a mechanism is based on a shared key between the parties, and in this setting is usually called a MAC, or Message Authentication Code. (Other terms include Integrity Check Value or Cryptographic Checksum). The sender appends to the data D an authentication tag computed as a function of the data and the shared key. At reception, the receiver recomputes the authentication tag on the received message using the shared key, and accepts the data as valid only if this value matches the tag attached to the received message.

...people seem agreed that hash function based constructions of MACs are worth having. The more difficult question is how best to do it.

The most common approach is to construct MACs from block ciphers like DES. Of such constructions the most popular is the CBC MAC. (Its security is analyzed in [4,12]). More recently, however, people have suggested that MACs might be constructed from cryptographic hash functions like MD5 and SHA-1. There are several good reasons to attempt this: In software these hash functions are significantly faster than DES; library code is widely and freely available; and there are no export restrictions on hash functions.

Thus people seem agreed that hash function based constructions of MACs are worth having. The more difficult question is how best to do it. Hash functions were not originally designed for message authentication. (One of many difficulties is that they are not even keyed primitives, i.e., do not accommodate naturally the notion of a secret key). Several constructions were proposed prior to HMAC, but they lacked a convincing security analysis.

The HMAC construction is intended to fill this gap. It has a performance which is essentially that of the underlying hash function. It uses the hash function in a black box way so that it can be implemented with available code, and also replacement of the hash function is easy should need of such a replacement arise due to security or performance reasons. Its main advantage, however, is that it can be proven secure provided the underlying hash function has some reasonable cryptographic strengths. The security features can be summarized like this: if HMAC fails to be a secure MAC, it means there are sufficient weaknesses in the underlying hash function that it needs to be dropped not only from this particular usage but also from a wide range of other popular usages to which it is now subject.

Several articles in the literature survey existing constructions, their properties, and some of their weaknesses, so we will not try to do this again here. In particular the reader is referred to Tsudik [17], who provides one of the earliest works on the subject; Kaliski and Robshaw who, in the first CryptoBytes [8], compare various possible constructions; updates appearing in succeeding issues of CryptoBytes; and Preneel and van Oorschot [12,13], who present a detailed description of the effect of birthday attacks on “iterated constructions” and also a new construction called MDx-MAC.

We now move on to discuss the HMAC construction, status, and rationale. For a complete description, implementation guidelines, and detailed analysis we refer the reader to [1,9].

Mihir Bellare is an assistant professor at UCSD. He can be contacted at mihir@cs.ucsd.edu. Ran Canetti is a post-doctoral fellow at MIT and can be contacted at canetti@theory.lcs.mit.edu. Hugo Krawczyk is a member of the Network Security Group at IBM. He can be contacted at hugo@watson.ibm.com.

HMAC

Let H be the hash function. For simplicity of description we may assume H to be MD5 [15] or SHA-1 [16]; however the construction and analysis can be applied to other functions as well (see below). H takes inputs of any length and produces an l-bit output (l=128 for MD5 and l=160 for SHA-1). Let Text denote the data to which the MAC function is to be applied and let K be the message authentication secret key shared by the two parties. (It should not be larger than 64 bytes, the size of a hashing block, and, if shorter, zeros are appended to bring its length to exactly 64 bytes.) We further define two fixed and different 64 byte strings ipad and opad as follows (the "i" and "o" are mnemonics for inner and outer):

ipad = the byte 0x36 repeated 64 times
opad = the byte 0x5C repeated 64 times.

The function HMAC takes the key K and Text, and produces:

$$HMAC_K(Text) = H(K \geq opad, H(K \geq ipad, Text))$$

Namely,

1. Append zeros to the end of K to create a 64 byte string
2. XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with ipad
3. Append the data stream Text to the 64 byte string resulting from step (2)
4. Apply H to the stream generated in step (3)
5. XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with opad
6. Append the H result from step (4) to the 64 byte string resulting from step (5)
7. Apply H to the stream generated in step (6) and output the result

The recommended length of the key is at least l bits. A longer key does not add significantly to the security of the function, although it may be advisable if the randomness of the key is considered weak.

HMAC optionally allows truncation of the final output say to 80 bits.

As a result we get a simple and efficient construction. The overall cost for authenticating a stream Text is close to that of hashing that stream, especially as Text gets large. Furthermore, the hashing of the padded keys can be precomputed for even improved efficiency.

Note HMAC uses the hash function H as a black box. No modifications to the code for H are required to implement HMAC. This makes it easy to use library code for H, and also makes it easy to replace a particular hash function, such as MD5, with another, such as SHA-1, should the need to do this arise.

HMAC uses the hash function H as a black box. No modifications to the code for H are required to implement HMAC.

HMAC was recently chosen as the mandatory-to-implement authentication transform for the Internet security protocols being designed by the IPSEC working group of the IETF (it replaces as a mandatory transform the one described in [10]). For this purpose HMAC is described in the Internet Draft [9], and in an upcoming RFC. Other Internet protocols are adopting HMAC as well (e.g., s-http [14], SSL [7]).

The rationale

We now briefly explain some of the rationale used in [1] to justify the HMAC construction.

As we indicated above, hash functions were not originally designed to be used for message authentication. In particular they are not keyed primitives, and it is not clear how best to "key" them. Thus, one ought to be quite careful in using hash functions to build MACs.

The standard approach to security evaluation is to look for attacks on a candidate MAC construction. When practical attacks can be found, their effect is certainly conclusive: the construction must be dropped. The difficulty is when attacks are not yet found. Should one adopt the construction? Not clear, because attacks might be found in the future.

The maxim that guided the HMAC construction was that an absence of attacks today does not imply security for the future. A better way must be found to justify the security of a construction before adopting it.

You can't make good wine from bad grapes: if no strengths are assumed of the hash function, we can't hope to justify any construction based on it. Accordingly it is appropriate to make some assumptions on the strength of the hash function.

You can't make good wine from bad grapes: if no strengths are assumed of the hash function, we can't hope to justify any construction based on it.

A well justified MAC construction, in our view, is one under which the security of the MAC can be related as closely as possible to the (assumed) security properties of the underlying hash function.

A well justified MAC construction, in our view, is one under which the security of the MAC can be related as closely as possible to the (assumed) security properties of the underlying hash function.

The assumptions on the security of the hash function should not be too strong, since after all not enough confidence has been gathered in current candidates (like MD5 or SHA-1). In fact, the weaker the assumed security properties of the hash function, the stronger the resultant MAC construction is.

We make assumptions that reflect the more standard existing usages of the hash function. The properties we require are mainly collision-freeness and some limited "unpredictability." What is shown is that if the hash function has these properties the MAC is secure; the only way the MAC could fail is if the hash function fails.

In fact the assumptions we make are in many ways weaker than standard ones. In particular we require only a weak form of collision-resistance. Thus it is possible that H is broken as a hash function (for example collisions are found) and yet HMAC based on H survives.

A closer look

Security of the MAC means security against forgery. The MAC is considered broken if an attacker, not having the key K, can find some text Text together with its correct MAC value $HMAC_K(Text)$. The attacker is assumed able to gather some number of example pairs of texts and their valid MACs by observing the traffic between the sender and the receiver. Indeed the adversary is even allowed a chosen message attack under which she can influence the choice of messages for which the sender computes MACs. Following [4,3] we quantify security in terms of the probability of successful forgery under such attacks.

The analysis of [1] applies to hash functions of the iterated type, a class that includes MD5 and SHA-1, and consists of hash functions built by iterating applications of a compression function f according to the procedure of Merkle [11] and Damgård [5]. (In this construction an l-bit initial variable IV is fixed, and the output of H on text x is computed by breaking x into 512-bit blocks and hashing in stages using f, in a simple way that the reader can find described in many places, e.g. [8].)

Roughly what [1] says is that an attacker who can forge the HMAC function can, with the same effort (time and collected information), break the underlying hash function in one of the following ways:

1. The attacker finds collisions in the hash function even when the IV is random and secret, or

2. The attacker is able to compute an output of the compression function even with an IV that is random, secret and unknown to the attacker. (That is, the attacker is successful in forging with respect to the application of the compression function secretly keyed and viewed as a MAC on fixed length messages.)

The feasibility of any of these attacks would contradict some of our basic assumptions about the cryptographic strength of these hash functions. Success in the first of the above attacks means success in finding collisions, the prevention of which is the main design goal of cryptographic hash functions, and thus can be assumed hard to do. But in fact, even more is true: success in the first attack above is even harder than finding collisions in the hash function, because finding collisions when the IV is secret (as is the case here) is far more difficult than finding collisions in the plain (fixed IV) hash function. This is because the former requires interaction with the legitimate user of the function (in order to generate pairs of input/outputs from the function), and disallows the parallelism of traditional birthday attacks. Thus, even if the hash function is not collision-free in the traditional sense, our schemes could be secure.

Some "randomness" of hash functions is assumed in their usage for key generation and as pseudo-random generators. (For example the designers of SHA-1 suggested that SHA-1 be used for this purpose [6].) Randomness of the function is also used as a design methodology towards achieving collision-resistance. The success of the second attack above would imply that these randomness properties of the hash functions are poor.

The analyses in [1] used to establish the above are exact (no asymptotics involved), consider generic rather than particular attacks, and establish a tight relationship between the securities.

Resistance to known attacks

As shown in [12,2], birthday attacks, that are the basis to finding collisions in cryptographic hash functions, can be applied to attack also keyed MAC schemes based on iterated functions (including also CBC-MAC, and other schemes). These attacks apply to most (or all) of the proposed hash-based constructions of MACs. In particular, they constitute the best known forgery attacks against the HMAC construction.

Consideration of these attacks is important since they strongly improve on naive exhaustive search attacks. However, their practical relevance against these functions is negligible given the typical hash lengths like 128 or 160. Indeed, these attacks require the collection of the MAC value (for a given key) on about $2^{l/2}$ messages (where l is the length of the hash output). For values of l • 128 the attack becomes totally infeasible. In contrast to the birthday attack on key-less hash functions, the new attacks require interaction with the key owner to produce the MAC values on a huge number of messages, and then allow for no parallelization. For example, when using MD5 such an attack would require the authentication of 2^{64} blocks (or 2^{73} bits) of data using the same key. On a 1 Gbit/sec communication link, one would need 250,000 years to process all the data required by such an attack. This is in sharp contrast to birthday attacks on key-less hash functions which allow for far more efficient and close-to-realistic attacks [18].

... birthday attacks ... constitute the best known forgery attacks against the HMAC construction. [...] However, for reasonable hash functions, their practical relevance is negligible.

References

[1] M. Bellare, R. Canetti and H. Krawczyk. Keying hash functions for message authentication. Advances in Cryptology – Crypto 96 Proceedings, Lecture Notes in Computer Science, N. Koblitz ed., Springer-Verlag, 1996.

[2] M. Bellare, R. Canetti and H. Krawczyk. Pseudorandom functions revisited: The cascade construction. Manuscript, April 1996.

[3] M. Bellare, R.Guérin and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. Advances in Cryptology – Crypto 95 Proceedings, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[4] M. Bellare, J. Kilian and P. Rogaway. The security of cipher block chaining. Advances in Cryptology – Crypto 94 Proceedings, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[5] I. Damgård. A design principle for hash functions. Advances in Cryptology – Crypto 89 Proceedings, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[6] National Institute for Standards and Technology. Digital Signature Standard (DSS). Federal Register, Vol. 56, No. 169, August, 1991.

[7] A.O. Freier, P. Karlton, and P.C. Kocher. The SSL Protocol – Version 3.0. Internet draft draft-freier-ssl-version3-01.txt, March 1996.

[8] B. Kaliski and M. Robshaw. Message Authentication with MD5. RSA Labs' CryptoBytes, Vol. 1 No. 1, Spring 1995.

[9] H. Krawczyk, M. Bellare and R. Canetti. HMAC-MD5: Keyed-MD5 for Message Authentication. Internet draft draft-ietf-ipsec-hmac-md5-txt.00, March 1996.

[10] P. Metzger and W. Simpson. IP Authentication using Keyed MD5. IETF Network Working Group, RFC 1828, August 1995.

[11] R. Merkle. One way hash functions and DES. Advances in Cryptology – Crypto 89 Proceedings, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989. (Based on unpublished paper from 1979 and his Ph.D thesis, Stanford, 1979).

[12] B. Preneel and P. van Oorschot. MD-x MAC and building fast MACs from hash functions. Advances in Cryptology – Crypto 95 Proceedings, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[13] B. Preneel and P. van Oorschot. On the security of two MAC algorithms. Advances in Cryptology – Eurocrypt 96 Proceedings, Lecture Notes in Computer Science, U. Maurer ed., Springer-Verlag, 1996.

[14] E. Rescorla and A. Schiffman. The Secure HyperText Transfer Protocol. Internet draft draft-ietf-wts-shttp-01.txt, February 1996.

[15] R. Rivest. The MD5 message-digest algorithm. IETF Network Working Group, RFC 1321, April 1992.

[16] FIPS 180-1, Secure Hash Standard. Federal Information Processing Standard (FIPS), Publication 180-1, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., April 1995.

[17] G. Tsudik. Message authentication with one-way hash functions. Proceedings of Infocom '92.

[18] P. van Oorschot and M. Wiener. Parallel Collision Search with Applications to Hash Functions and Discrete Logarithms. Proceedings of the 2nd ACM Conf., Computer and Communications Security, Fairfax, VA, November 1994.

ANNOUNCEMENTS

The 1996 RSA Laboratories Seminar Series

RSA Laboratories is pleased to announce preliminary details of the 1996 RSA Laboratories Seminar Series which will be held on August 15 and 16 in Palo Alto, California. (The annual Crypto conference will be held the following week in Santa Barbara, California.)

The aim of the Seminar Series, as in previous years, is to provide a bridge between academics who devise and analyze cryptographic technology and practitioners who are involved in actually implementing and integrating this technology into products.

This year we intend to use the Seminar Series to provide a snapshot of the state of cryptographic research and its relation to the practical world. Invited speakers will address issues in a variety of cryptographic fields and provide an assessment of what the future might hold. Sessions will be informal and relaxed and attendees, who might perhaps see a different side to cryptographic technology, will be encouraged to ask questions and comment on the direction of current research.

It is not often clear how much interaction takes place across what can sometimes be a significant division between academic specialists and the practitioners of cryptographic techniques. Our aim is to provide a forum within which both parties involved in the development of cryptography can meet to exchange ideas and views on both current knowledge and the potential for future advances.

More complete details about the 1996 RSA Laboratories Seminar Series will be available soon and made available on the World-Wide Web at http://www.rsa.com/rsalabs/. In the meantime more information can be obtained by contacting RSA Laboratories by any of the means given on the inside front cover.

For contact and distribution information, see page 2 of this newsletter.

RSA LABORATORIES
CRYPTOGRAPHIC RESEARCH AND CONSULTATION
100 MARINE PARKWAY
REDWOOD CITY
CA. 94065-1031
TEL 415/595-7703
FAX 415/595-4126
rsa-labs@rsa.com