COMBINING AUTHENTICATION AND LIGHT-WEIGHT PAYMENT FOR ACTIVE NETWORKS

Rüdiger Weis
cryptolabs Amsterdam
convergence integrated media GmbH, Berlin, San Francisco, Amsterdam
ruedi@cryptolabs.org

Wolfgang Effelsberg
Praktische Informatik IV
University of Mannheim, 68131 Mannheim, Germany
effelsberg@pi4.informatik.uni-mannheim.de

Stefan Lucks*
Theoretische Informatik
University of Mannheim, 68131 Mannheim, Germany
lucks@th.informatik.uni-mannheim.de

Keywords: Active Networks, Payment, Authentification, Combined Schemes

Abstract Security functions are of critical importance for the acceptance of Active Networks in practice: network nodes must be protected from malicious code, and they should account for the cost of executing code; this also helps to prevent denial-of-service attacks. For the payment function code packets must carry some form of light-weight electronic cash. Cryptographic schemes can be used to solve both the security and the payment/resource management problem. In this paper we propose to *combine* cryptographic algorithms in order to solve both problems in an integrated way. Our scheme is secure, light-weight and efficient: It saves space in the packet headers, and the security is higher than that of separate algorithms for authentication and cost accounting.

*Supported by the Deutsche Forschungsgemeinschaft (DFG) grant KR 1521/3-1.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35522-1_37

H. R. van As (ed.), *Telecommunication Network Intelligence*

1. INTRODUCTION

Active Networks are considered to be especially useful for the rapid development of new services in a network. It is a key characteristic of Active Networks that packets carry not only data but also code. This code is written to be executed on internal nodes of the network. The internal nodes are thus expected to execute "foreign" code, i.e. code originating at remote sites. They must have means to execute authorized code only in order to prevent damage to local resources and to other network users.

Another problem with Active Networks is the fact that the execution of "foreign" code consumes local resources in the internal nodes. If there is no accounting in the internal nodes one packet stream could waste considerable processing power within the network; fairness between streams could not be enforced. Also denial-of-service attacks could not be prevented, i.e. an arriving packet could run into a code loop and block a node[1].

We propose to use a micro-payment protocol, initially developed for electronic commerce between humans, for this purpose. It turns out that the same cryptographic building blocks can be used for code authentication and for micro-payment[2]. We therefore propose to *combine the two functions* for Active Networks.

Since there are many different models for Active Networks in the literature we define the model on which our work is based in Section 2. In Section 3 we give a short overview of light-weight payment protocols. Section 4 presents a scheme to combine code error detection with the payment function. In Section 5 we introduce a slightly more complex scheme that combines code authentication with payment. Section 6 formally proves the security of our combined schemes. Section 7 discusses some implementation aspects, and Sections 8 concludes the paper.

2. OUR ACTIVE NETWORK MODEL

There are two fundamental ways of combining code and data in Active Networks:

- In the *immediate one-time execution* architecture each packet carries its own code; the code is extracted in each internal network node and executed locally. This architecture is very similar to Mobile Agent systems.

- In the *demand loading* architecture the code is initially loaded into an internal node when it is requested by a packet; it can remain

there and be executed many times, typically when processing the subsequent packets of the same data stream.

In this paper we only consider the immediate one-time execution architecture where the code and data of each packet are self-contained. We call such a packet an *ANpacket.*

New ANpackets are generated by *client nodes.* Each client node is attached to a node of the active network, an *internal node.* The internal nodes are interconnected by secure links. All internal nodes are *trusted nodes*; our scheme does not provide protection of one internal node from malicious other internal nodes.

The purpose of our scheme is to

- authenticate ANpackets in the internal nodes (i.e., verify the client node they come from),
- provide a payment scheme by which the ANPackets pay for the services of the internal nodes. This payment scheme also helps to prevent denial-of-service attacks.

The ANpackets travel along an arbitrary path of internal nodes which provide code execution services. In order to distribute cryptographic keys and payment information efficiently we also assume that an efficient and reliable broadcast mechanism exists within the internal network. Our architectural model is illustrated in Figure 1.

A real-world example corresponding to our model could be a Virtual Private Network (VPN) where the internal network runs on trusted hosts interconnected by secure links. Another example would be an Internet Service Provider (ISP) where the inner network is a set of routers that use ANpackets for signalling or resource reservation purposes (but probably not in the main data path). It is typically assumed that one IP router can trust another IP router within the network of an ISP.

Without a payment function each ANpacket could execute as much code as it wishes on an internal node. This can lead to very unfair behaviour: a source node can inject a valid packet into the network that uses up most of the processing power of an internal node. Also, denial-of-service attacks are easy: a source node generates a packet with a code loop, to be executed on an internal node, and that node will become unavailable to other packets streams.

We therefore propose a resource management scheme based on micro-payment. Each packet carries a chain of electronic coins. These coins pay for local services on internal nodes. When the money is used up the ANpacket is discarded by the internal node. The generation and

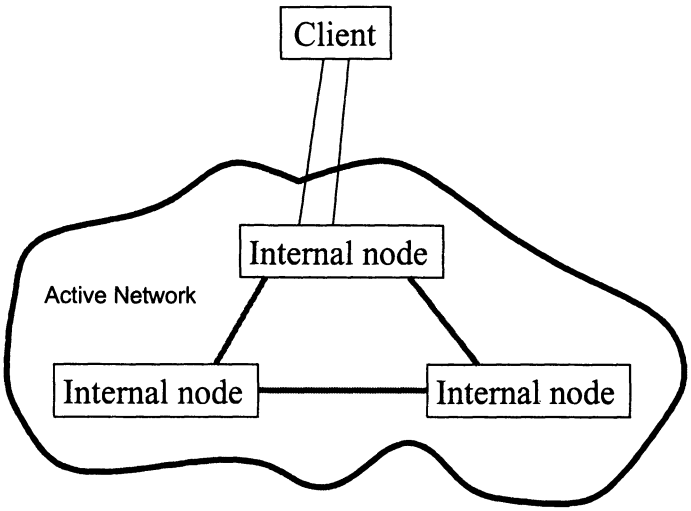

Figure 1 Architectural Model of the Active Network

consumption of coins is done with an efficient, light-weight payment protocol which we introduce in the next section.

3. A LIGHT-WEIGHT PAYMENT PROTOCOL

In the last years several light-weight payment schemes have been developed. Most of them are based on cryptographic hash functions. The main reason for this approach is the reduction of public key operations to gain better performance. As Rivest and Shamir have pointed out hash functions are roughly 1,000 times faster than RSA signature verification (with a small public exponent), and about 10,000 times faster than RSA signature generation [RiSh96].

3.1. THE PAYWORD SCHEME

An especially interesting scheme is the Payword scheme. It is fast, easy to implement and provides a high degree of security because of the use of simple operations. It is a payment scheme based on chains of hash values. It was developed by Ron Rivest and Adi Shamir [RiSh96]. Similar chains have been previously proposed by Lamport [Lamp81] and Haller in S/Key [Hall94], for access control (see also [OPIE]). The idea

for such micropayments has also been independently discovered by Anderson et al. [AMS95] and Pedersen [Pede95].

With $\mathcal{H}$ we denote a cryptographically strong hash function, such as SHA-1 [FIPS180] or RIPE-MD160 [DBP96]. The important property of $\mathcal{H}$ is its *one-wayness* and *collision-resistance*. Note that the "main security" of the scheme is based on the non-invertibility of a cryptographic hash function, a weak cryptographic assumption.

Generating Paywords. In the first step the user U has to establish an account with the broker B. U creates a payword chain in reverse order by picking the last chain link w_n at random, and then computing

$$w_i = \mathcal{H}(w_{i+1}) \text{ for } i = n-1, \ldots, 1$$

We designate w_0 as the root of the payword chain. The broker B signs a digital certificate containing the broker's name, the user's name and IP-address, the user's public key, the expiration date, the root w_0 of the payword chain, the length of the payword chain and other information. This is illustrated in Figure 2.

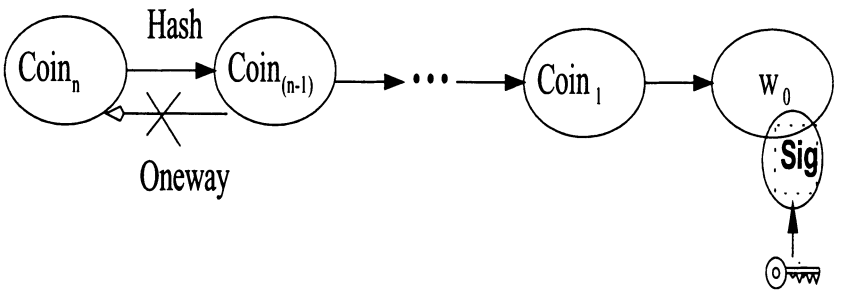

Figure 2 Hash chain

Performing a Payment. The i-th coin (for $i = 1, 2, \ldots$) from U to a vendor V consists of the pair (w_i, i). The vendor can verify the coin by checking

$$\mathcal{H}(w_i) \stackrel{?}{=} w_{i-1}.$$

So each payment requires no additional calculations by user U, and only a single hash operation by payment receiver V.

3.2. THE PAYWORD SCHEME IN THE ACTIVE NETWORK CONTEXT

The client is playing the role of the user U and the internal nodes play the role of the vendors V. The network owner might be the coin broker B. Before processing begins the payment root w_0 is broadcast to all internal nodes. The initial; distribution of "money" to the client nodes is beyond the scope of this paper.

An additional feature of our approach is that we can use the payment scheme for internal accounting in a very simple and natural manner. The Payword scheme supports variable amount payments without increasing the coin size. We can simply skip chain links to pay a higher amount, in multiples of the coin value.

Suppose that each coin is worth one cent. If U is given w_4 instead of w_1, the payment has the value of 4 cents. Now the first internal node has to perform four hash calculations to check the validity of the w_4 coin. He can "consume" one coin by using the "intermediate value" w_3 as the start of a new payment chain for the next internal node along the route (see Figure 3).

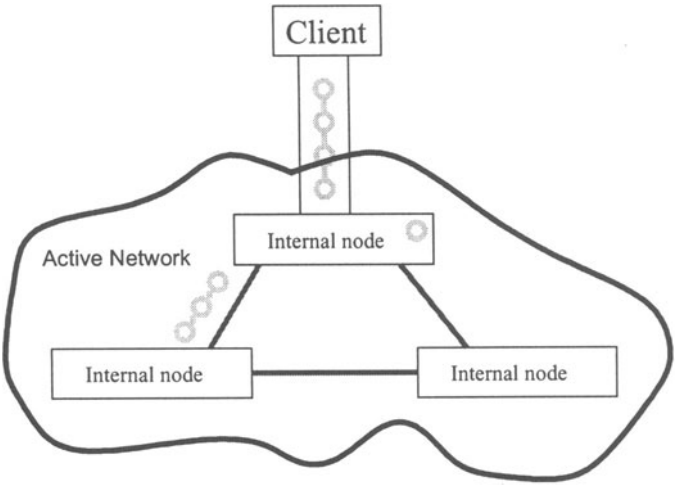

Figure 3 Payword Scheme for Active Networks

4. COMBINING ERROR DETECTION AND PAYMENT

We can check the *integrity* of an ANpacket by using a cryptographic checksum. Let $\mathcal{H}$ be a collision-resistant hash function and *coin* a coin of the payment scheme.

The payment scheme uses the boolean function $ValidateCoin$ to check the validity of a coin.

We define a function $\mathcal{HC}$ which combines both with a bitwise XOR

$$\mathcal{HC}(\cdot) \mapsto \mathcal{H}(\cdot) \oplus coin.$$

Now we can compute a "hash&coin" check field by

$$HCfield := \mathcal{HC}(ANpacket).$$

This is illustrated in Figure 4. The client node concatenates the contents

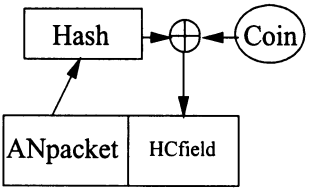

Figure 4 The Error Detection and Payment Scheme

of the packet with the HCfield

$$Send(ANpacket||HCfield)$$

An internal network node can check the validity of a packet by the *Check* function using the $ValidateCoin$ function of the payment scheme. If we use the Payword protocol this implies only one other hash computation:

$$Check(ANpacket||HCfield) := ValidateCoin(HCfield \oplus \mathcal{H}(ANpacket))$$

If there was a modification of the ANpacket in transit, or the coin is invalid, the packet will be marked as "invalid" and discarded.

The possibility that two "false" values compensate each other can be neglected if we choose sufficient lengths for the hash code and the coins. We recommend to use $b := 160$ bit, which is the output-length of SHA-1 and RIPEMD-160.

Man-in-the-Middle Attack

Because of the fact that in our first scheme no secret information is used to calculate the checksum, the scheme can only be secure against a *passive attacker*. A passive attacker can read every protocol message, but is not able to modify, generate or destroy messages.

An active attacker can intercept an $HCfield$ and calculate the coin by using the public hash function $\mathcal{H}$:

$$HCfield \oplus \mathcal{H}(ANpacket) = Coin$$

Now he can use the coin for his own packet. This motivates us to provide a modified scheme for source authentification.

5. COMBINING AUTHENTICATION AND PAYMENT

We can check the *authenticity* of an ANpacket by using a Message Authentication Code (MAC) instead of a public hash function. This provides security against an active attacker. Note that there must be a secure channel for distributing the secret MAC key.

The Authentification and Payment Scheme is very similar to the scheme of the previous section. Let $\mathcal{MAC}$ be the Message Authentication Code and *coin* a coin of the payment scheme.

We define a function $\mathcal{AC}$ which combines the MAC and the Coin with a bitwise XOR

$$\mathcal{AC}(\cdot) := \mathcal{MAC}_K(\cdot) \oplus Coin$$

We then calculate the "Authentication&Coin" $ACfield$

$$ACfield := \mathcal{AC}(ANpacket).$$

The client node concatenates the contents of the packet with the AC field

$$Send(ANpacket||ACfield)$$

The scheme is illustrated in Figure 5. An internal node can check the validity of a packet by calculating the MAC and calling the $ValidateCoin$ function of the payment scheme:

$$Check(ACfield) := ValidateCoin(ACfield \oplus \mathrm{MAC}_K(ANpacket))$$

If there was a modification of the ANpacket in transit or the coin is invalid, the packet will be marked as "invalid" and discarded.

The possibility that two "false" values compensate each other can again be neglected if we choose a sufficient length for the MAC and the coins. We recommend to use $b := 160$ bit.

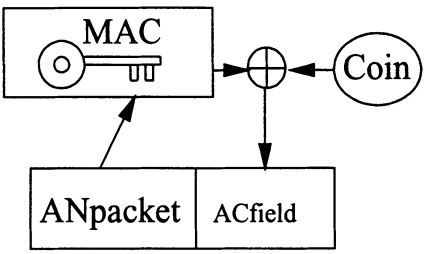

Figure 5 The Combined Authentication and Payment Scheme

6. SECURITY ANALYSIS

Our fundamental idea is to combine two cryptographic outputs using an algebraic group operation – such as the bit-wise XOR "$\oplus$" – in order to save space in the packet header. At a first look, the security of the combined scheme appears questionable. Intuitively, one would expect the combined scheme to be insecure if either of its components is insecure. In this section we give evidence that this intuition is wrong, and the combined scheme is secure if at least one of its components is secure.

In a formal model, we provide a proof of security for the combined scheme. For the sake of space, we concentrate on the combination of authentication and payment, as described in Section 5.

By "proof of security" we mean a formal proof that the combined scheme can only be broken, if *both* the authentication *and* the payment scheme can be broken. As a first step, we must define what it means if a scheme "can be broken".

Similar ideas to ours have been used in the context of cascading ciphers [MaMa93] and in the design of the RIPEMD-160 hash function [DBP96].

6.1. DEFINITIONS

Our scheme combines an authentication value, a "Message Authentication Code" (MAC), and a payment value, a "coin". We first define the security requirements for MACs and coins, then we define the security requirement for our scheme.

Adaptive Chosen Message Attack

A MAC depends on a secret key: everyone knowing the secret key K can compute the MAC

$$m = \mathcal{MAC}_K(x)$$

for the input x. On the other hand, it must be infeasible for any adversary to forge a MAC, i.e., to compute m without knowing K. While the key K is used by the legitimate users, the adversary may learn some pairs

$$(x_1, \mathcal{MAC}_K(x_1)), (x_2, \mathcal{MAC}_K(x_2)), \ldots,$$

and, in the worst case, may even be able to choose the values x_i. This is a so-called *Adaptive Chosen Message Attack.* Being quite pessimistic, we consider this type of attack to be possible. For example the adversary might modify the traffic on the link from the client to the first internal node.

In our formal model, we say that the adversary chooses x_i and learns the value $m_i = \mathcal{MAC}_K(x_i)$ from a "MAC-oracle". Then, the adversary chooses x_{i+1} and learns $m_{i+1} = \mathcal{MAC}_K(x_{i+1})$ from the oracle.

q-forgery-secure MAC Scheme

A MAC is *q-forgery-secure* if after choosing q values $x_1, \ldots, x_q$ and learning the MACs

$$m_1, \ldots, m_q \text{ with } m_i := \mathcal{MAC}_K(x_i)$$

from the oracle, it is infeasible to forge a MAC, i.e., to find a pair (x_0, m_0) with

$$m_0 = \mathcal{MAC}_K(x_0) \text{ and } x_0 \notin \{x_1, \ldots, x_q\}$$

without asking the oracle a $(q+1)$-st time.

Our payment scheme depends on sending coins c_q, c_{q-1}, ... to the receiver. A payment scheme is secure, if the adversary cannot forge coins, except when these are created by herself. In contrast to MACs, there is nothing to choose for the adversary.

q-forgery-secure Payment Scheme

A payment scheme is *q-forgery-secure* if after having learned up to q valid coins c_s, c_{s-1}, ..., c_{s-q+1}, it is infeasible for the adversary to produce another valid coin c_{s-q}. For the sake of simplicity, we assume the coin c_{i-1} to be uniquely defined by the coin c_i.

q-forgery-secure Combined Scheme

Similar to the above, we now define the security of our combined scheme for authentication and payment. We allow the adversary to choose a packet p_i, to learn the corresponding value a_i from an "AC-oracle", then to choose a packet p_{i+1}, to learn the corresponding value

$a_{i+1} \ldots$ with

$$a_i = \mathcal{AC}(p_i) = \mathcal{MAC}_K(p_i) \oplus c_{s-i}.$$

Here, c_s, c_{s-1}, ... are our coins.

The scheme is *q-forgery-secure*, if after having chosen q values x_i and learned q AC-values a_i, it is infeasible for the adversary to find another pair (p_0, a_0) with

$$p_0 \notin \{p_1, \ldots, p_q\} \text{ and } a_0 = \mathcal{AC}(p_0).$$

on her own.

6.2. PROOF OF SECURITY

In this section, we give the proof advertised above.

Lemma 1: If the MAC is q-forgery-secure, then the composed scheme is q-forgery-secure.

Sketch of proof. An outline of the proof is as follows: assume the existence of an algorithm A to break the composed scheme. We describe an algorithm A* to forge MACs, using A as a "subroutine". The running time for A* is the running time for A plus a moderate amount of extra work. Essentially, A* computes the responses to A's q oracle queries:

1 A* creates $q + 1$ coins $c_q, \ldots, c_0$.

2 A* runs A.

Every time A chooses p_i and asks the AC-oracle for $a_i = \mathcal{AC}(p_i)$, A* asks the MAC-oracle for $m_i = \mathcal{MAC}_K(p_i)$ and responds with $a_i = m_i \oplus c_{q-i+1}$.

3 Finally, A produces an output (p_0, a_0) with $p_0 \notin \{p_1, \ldots, p_q\}$ and $a_0 = \mathcal{AC}(p_0)$. Then A*'s output is $m_0 = a_0 \oplus c_0$.

The value m_0 is a successfully forged MAC, i.e., $p_0 \notin \{p_1, \ldots, p_q\}$ and $m_0 = \mathcal{MAC}_K(p_0)$.
q.e.d.

Lemma 2: If the payment scheme is q-forgery-secure, then the composed scheme is q-forgery-secure.

Sketch of proof. Similarly to the proof of Lemma 1, we assume the existence of an algorithm A to break the composed scheme and describe an Algorithm A** using A as a "subroutine" to break the Payment scheme:

1 A** chooses a key K for the MAC scheme.

2 A** runs A.

Every time, A chooses p_i and asks the AC-oracle for $a_i = \mathcal{AC}(p_i)$, A** asks for the coin c_{q-i+1}, computes $m_i = \mathcal{MAC}_K(p_i)$ and responds with $a_i = m_i \oplus c_{q-i+1}$.

3 Finally, A produces an output (p_0, a_0) with $p_0 \notin \{p_1, \ldots, p_q\}$ and $a_0 = \mathcal{AC}(p_0)$. Then A** computes $m_0 = \mathcal{MAC}_K(p_0)$ and outputs $c_0 = a_0 \oplus m_0$.

The value c_0 is a successfully forged coin.
q.e.d.

Theorem: If either the MAC scheme or the payment scheme is q-forgery-secure, then the composed scheme is q-forgery-secure.

Proof: This follows directly from Lemma 1 and Lemma 2. q.e.d.

In other words, our scheme is secure in the above sense if either of its two components is secure, even if the other component is insecure.

6.3. REMARKS

In cryptography, one often requires MACs to be even stronger than forgery secure: If the adversary produces $q+1$ MAC inputs $x_1, \ldots, x_q$, and x_0 and learns the responses $m_i = \mathcal{MAC}_K(x_i)$ for $i \in \{1, \ldots, q\}$ and then learns another value m_0 which is either (a) $m_0 = \mathcal{MAC}_K(x_0)$ or a (b) random value m_0, then the MAC scheme is *q-distinguishing-secure* if it is infeasible for the adversary to distinguish between options (a) and (b).

If a MAC is distinguishing-secure, there is no way for the adversary to derive any useful information from the MAC. Similarly, we might define the distinguishing-security of payment schemes and composed authentication/payment schemes.

In this paper, we concentrate on forgery security because we believe that it meets most practical security demands for Active Networks. Note that hash-chain based payment schemes, as suggested in this paper, actually are distinguishing-*in*secure: given the coin c_i, everyone can verify $c_{i+1} = \mathcal{H}(c_i)$ and hence check a coin's authenticity.

Based on the famous work of Maurer and Massey [MaMa93], one can provide a formal framework similar to ours to prove the security of our composed schemes: if either of the two components behaves like a random function, then so does the composes scheme. But note that being random is a much stronger requirement than just being forgery secure. The formal treatment presented above depends on the much weaker assumption of forgery security and hence indicates an improved

margin of safety, compared to a formal treatment based on requiring pseudorandomness.

We stress that the proof of security is still applicable if "$\oplus$" is replaced by any group operation, even a non-commutative one.

7. IMPLEMENTATION

In this section we discuss some implementation aspects. Based on the security proof, we suggest to use different realisations of the MAC scheme and the Payword scheme. First we discuss the HMAC construction. Then we take a short look at hash functions not based on MD4. Finally we suggest to use "salting" to improve the security of hash chains.

7.1. THE HMAC CONSTRUCTION

To save code and to be able to use standard cryptographic libraries, we suggest to use a Message Authentification Code (MAC) based on the same $\mathcal{H}$ as the payword chain. HMAC [BCK96] uses a cryptographic hash function as a black box:

$$\mathrm{HMAC}_K(x) = \mathcal{H}(\bar{K} \oplus \mathsf{opad} || \mathcal{H}(\bar{K} \oplus \mathsf{ipad} || x))$$

with ipad := 0x36 repeated 64 times and opad := 0x5C repeated 64 times, $\bar{K}$ is generated by appending zeros to the end of K to create a 512 bit string.

This approach has several advantages. Cryptographic hash functions have been well studied, and are usually faster than encryption algorithms. In many countries, it is easier to export or import an authentication tool, such as a signature system, than to export or import a secure encrypting system.

Security of HMAC. In [BCK96] it was proven that HMAC provides security against collision and forging attacks with only weak assumptions on the underlying hash function.

This leaves an additional margin of security: even if some weakness in the hash function $\mathcal{H}$ (e.g. MD5) is found, the MAC based on $\mathcal{H}$ may still be secure. For example a collision of hash function means finding a collision with a fixed Initial Vector (IV) and known output. If an attacker wants to find a collisions in HMAC, she must find a collision in the underlying hash function even when the IV is random and secret, and the hash value is not explicitly known.

HMAC based on SHA-1 or RIPEMD-160 provides a 160-bit output. So even birthday attacks which need 2^{80} operations seem to be infeasible.

7.2. HASH FUNCTIONS WITH DIFFERENT DESIGN

It follows directly from the security proof in Section 6 that our scheme is secure if *one* building block is secure. So it might be a good idea to use building blocks based on *different* design. If a weakness is found in one design, we still have a security reserve. An attacker will have to break *both*. For example we can use RIPE-MD based HMAC and a hash chain for payment based on Tiger.

Tiger. Tiger [AnBi96] is a new fast iterative Hashfunction designed by Ross Anderson and Eli Biham. In contrast to MD5, RIPE-MD, SHA and HAVAL it is *not* based on MD4. Tiger uses a big 8×64-S-Box to gain a fast avalanche. Because of the extensive use of 64-bit operations Tiger is as fast as SHA-1 on 32-bit processors, and about three times faster on modern 64-bit processors. The algorithm produces a 192 bit output, which can be easily reduced to 160 or 128 bit by taking the first 160 respectively 128 bit from the 192 bit output.

7.3. SALTING HASH CHAINS

Salting is an inexpensive way to make dictionary attacks much more difficult. Simply replace $\mathcal{H}(\cdot)$ by

$$\mathcal{H}_s(\cdot) = \mathcal{H}(s||\cdot),$$

where s is a "salt" (random value) which can be specified in the commitment [RiSh96].

If we use a standard iterative hash function, the input (160 bit in our case) is padded to a longer bit string (typically $n \cdot 512$ bit). So we have no relevant performance disadvantage.

8. CONCLUSION

In Active Networks the security of packets and of internal network nodes is a vital necessity. Also, managing the active nodes' resources in the presence of unpredictable incoming code packets is quite difficult. Micropayment schemes help to manage node resources effectively and in a fair manner.

In our schemes, the same header field is used for both authentication and payment purposes. This saves space in the packet header. We have presented a scheme which combined bit error detection with micropayment and a scheme which combined authentication with micropayment.

We also gave a formal justification that our schemes are sound, i.e., secure if the underlying building blocks are secure. Our proof is very

strong: even if one of our two components is broken, the other one only has to satisfy rather weak security requirements.

9. ACKNOWLEDGEMENTS

The authors want to thank the anonymous reviewers and Martin Mauve for useful hints to improve the paper.

Notes

1. Several authors have proposed the use of light-weight accounting schemes to control resource consumption; see for example [Tsch97].

2. Researchers at [IBM] have proposed a protocol family called KryptoKnight that is also based on heavy use of hash functions in order to avoid dedicated encryption functions for authentification and key distribution [BGH+92, MTVZ92, BGH+95].

References

[AnBi96] Anderson, R.,Biham, E., "Tiger: A Fast New Hash Function", Fast Software Encryption 3, LNCS 1039, 1996, pp. 89-98.

[AMS95] Anderson, R., Manifavas, H., Sutherland, C., "A practical electronic cash system", 1995.

[BCK96] Bellare, M., Canetti, R., Krawczyk, H, "Keying hash functions for message authentication", Advances in Cryptology - Crypto 96 Proceedings, Springer, 1996.

[BETT98] Banchs, A., Effelsberg, W., Tschudin, C., Turau, V.: "Active Multicasting of Multimedia Streams", Proc. IEEE Local Computer Networks Conference LCN'98, Lowell, MA, October 1998, pp. 150-159

[BGH+92] Bird, R., Gopal, I., Herzberg, A., Janson, P., Kutten, S., Molva, R., Yung, M., "Systematic Design of a Family of Two-Party Athentification Protocols", CRYPTO'91, Springer LNCS, 1992, pp. 44–61.

[BGH+95] Bird, R., Gopal, I., Herzberg, A., Janson, P., Kutten, S., Molva, R., Yung, M., "The KryptoKnight family of light-weight protocols for authentication and key distribution", IEEE/ACM Trans. Networking 3, 1, Feb. 1995, pp. 31 - 41.

[DBP96] Dobbertin, H., Bosselaers, A., Preneel, B., "RIPEMD–160, a strengthened version of RIPEMD", Proc. Fast Software Encryption (ed. D. Gollmann), LNCS 1039, Springer, 1996, pp. 71–82.

[FIPS180] NIST, "Secure Hash Standard", Washington D.C., April 1995.

[Hall94] Haller, N., "The S/KEY One–Time Password System", Proc. of the ISOC Symposium on Network and Distributed System Security, San Diego, CA, February 1994.

[HSW95] Hauser, R., Steiner, M., Waidner, M., "Micro-Payments based on iKP", December 17, 1995.

[IBM] IBM Applied Computer Science, KryptoKnight
http://www.zurich.ibm.com/Technology/Security/extern/kryptoknight/

[Lamp81] Lamport, L., "Password Authentication with Insecure Communication", Communications of the ACM 24(11), November 1981.

[MaMa93] Maurer, U., Massey, J., "Cascade ciphers: the importance of being first" Journal of Cryptology. Vol. 6. Nr. 1., 1993, pp 89-105.

[MTVZ92] Molva, R., Tsudik, G., Van Herrweghen, E., Zati, S., "KryptoKnight Authentification and Key Distribution System", Europeam Symposium on Research in Computer Security (ESORICS'92), Toulouse, 1992.

[OPIE] One-time Password in everything, US Naval Research Laboratory,
ftp://ftp.nr1.navy.mil/pub/security/opie

[Pede95] Pedersen, T., "Electronic payments of small amounts", Technical Report DAIMI PB-495, Aarhus University, Computer Science Department, August 1995.

[RiSh96] Rivest, R., Shamir, A., "Payword and Micromint", Security Protocols, Springer LNCS 1189, 1997, pp. 1-18.
http://theory.lcs.mit.edu/~rivest/RivestShamir-mpay.ps

[Tsch97] Tschudin, C., "Funny Money Arbitrage for Mobile Code", Dartmouth Workshop on Transportable Agents, Extended Abstract, August 1997.
http://www.icsi.berkeley.edu/~tschudin/dart97.txt

[WGT98] Wetherall, D., Guttag, J., Tennenhouse, D., "ANTS: A Toolkit for Building and Dynamically Developing Network Protocols", Proc. OpenArch 98, San Francisco, April 1998, pp 117-129.
http://www.tns.lcs.mit.edu/publications/openarch98.html