A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication

Pankaj Rohatgi
I.B.M. T.J.Watson Research Center,
P.O.Box 704,
Yorktown Heights, NY 10598, U.S.A.
Email: rohatgi@watson.ibm.com

Abstract

This paper proposes a compact and fast hybrid signature scheme that can be used to solve the problem of packet source authentication for multicast. This scheme can be viewed as an improvement to
off-line/on-line signature schemes, in that the signature size overhead is much smaller. Since this is a generic technique, it should have applications to several other practical problems as well.

1 Introduction

1.1 The Multicast Source Authentication Problem

Packet Source Authentication, i.e., authenticating the source of a received packet, is a fundamental security issue in any networking protocol. In the case of Unicast, this problem has been solved (e.g., in IPSEC) by the use of message authentication codes (MACs). However, the MAC approach is inadequate in a multicast setting. This is because a MAC is based on a shared secret key among participants and MACs can be both generated and verified by anyone having access to the key. If a MAC was used in a multicast setting then all authorized receivers would have to know the secret key in order to authenticate received packets and therefore any authorized receiver could inject packets with valid MACs and masquerade as a sender in order to attack other receivers. This problem does not arise in unicast where there is only one receiver.

The best approach that can solve this problem from a security perspective would be for the sender to digitally sign each packet. Any recipient could then authenticate signed packets without gaining the ability to forge packets. The main problem with this approach is performance. Public key signatures using acceptable algorithms and key-lengths are very compute-intensive to generate or to verify or both. As an example, even a high end workstation such as a 200 MHz PPC 604e based RS/6000 can generate only 40 or so 1024 bit RSA signatures per second. Clearly, some multicast applications could require a packet rate far exceeding 40 packets/s and most applications which require less throughput, nevertheless cannot afford to devote a large fraction of CPU cycles doing signatures. Also, if a server is serving multicast data to several multicast groups, the number of public-key signatures that it can perform per second per group will be severely limited. Thus this solution is not practically feasible.

1.2 Prior/Related work and its shortcomings

One possible solution to this problem would be to relax the security requirements for authentication. If a certain amount of risk is acceptable, then one could use less-studied signature algorithms. This approach could provide fast yet secure authentication provided the underlying cryptographic assumptions are valid. However, if these assumptions turn out to be invalid, these schemes may be completely open to compromise. Another efficient approach to multicast packet authentication when the security requirement is relaxed was proposed in [5] where the concept of "asymmetric MACs" was introduced. The basic idea in this approach is that the sender knows several secret MAC keys and these keys are shared with the recipients in such a manner so as to maintain several properties of the subsets of keys held by the recipients. For example, one such property could be that no collection of w receivers should know all the keys known by any other receiver. The server can authenticate a message by computing MACs using all its secret keys and appending all these MACs to the message (another variant uses more MACs with each MAC being only 1-bit long). This collection of MACs is known as an asymmetric MAC. Each recipient can verify the parts of the asymmetric MAC for which it knows the secret keys and if all these MACs verify then the receiver accepts the message as genuine. Note that a receiver, by itself cannot forge an asymmetric MAC since it does not know all the keys of a sender or even all the keys known to some other recipient. From the property of the subsets described above, even w receivers cannot collude to forge an asymmetric MAC to fool some other recipient. However once there are more than w colluders the security of the scheme could break down and once there are sufficient colluders to know all the sender keys then the scheme breaks down completely. It is easy to see that the number of MAC's computed by the sender has to be a linear function of the number of colluders that the scheme is supposed to be resilient against. Therefore, even though this scheme is useful in many scenarios, e.g., when groups are small and problems of collusion can be controlled, it does not work well in scenarios where the multicast group is very large and large collusions are likely to occur or difficult to detect. The two big advantages with this approach however

is that it can employ well-studied and cryptographically secure MACing schemes and remain secure till the limit on the number of colluders is reached and also for small groups or groups with small number of expected colluders the scheme is very efficient in terms of CPU usage and size overhead.

If reliability of transmission was not an issue, there is another approach known as stream signing [7] that could be used to sign the multicast packets efficiently and provide the security guarantees associated with digital signatures. In this approach only one regular signature is transmitted at the beginning of a stream and each packet either contains a cryptographic hash of the next packet in the stream or a 1-time public key using which the 1-time signature on the next packet can be verified. However, this approach, as specified cannot tolerate a lost packet, since the information needed to authenticate future packets will be lost. While this may not be an issue in reliable internet protocols (such as those based on TCP/IP), it is a major issue for many multicast applications such as those involving audio/video delivery and multicast applications which use UDP over IP-Multicast which is inherently unreliable. In particular, the lack of any standard for reliable multicast over IP means that for the time being UDP over IP-Multicast remains the only non-proprietary method available for using multicast over the internet. Another problem with [7] is that if the stream being sent is not known to the sender in advance then the sender needs to embed 1-time keys and signatures into the packet stream. These keys and signatures are fairly large and can result in a substantial space overhead within each packet. While some of the overhead can be reduced if the sender is allowed to delay outgoing packets, delay is not a viable option for peer-to-peer interactive multicast applications such as distributed simulations/gaming. The results presented in this paper can also be used to substantially reduce the space overhead of this scheme, without introducing a delay, and this combination may be a very good solution to the authentication problem for reliable multicast.

In [11], a different approach was proposed for packet authentication when a sender is allowed to delay and group together several consecutive packets. Essentially, this approach forms an authentication tree [12] from packets collected during a time-interval and signs the root of the authentication tree. Then each packet is augmented by the signature on the root and ancillary information which includes hashes on the logarithmically many nodes on the authentication tree. This allows packet can be individually verified. This approach is quite effective in client-server scenarios where the server is dedicated to serving a very small number of multicast flows, each with reasonably smooth flow rates and strictly enforced bounds on processor loading. This approach too has several practical drawbacks. Firstly, as discussed earlier, delaying and grouping of sender's packets is not possible for highly interactive peer-to-peer multicast applications such as distributed simulations/gaming. Secondly, there is the problem of serving multiple multicast flows. This is best illustrated by the following example. Suppose a server only has enough spare cycles to perform 10 public key operations per second. Using scheme in [11], such a server could easily serve a single smooth flow of hundreds of authenticated packets per second with only a minor delay of a fraction of a second. However, if the same server was required to send only 50 different low bandwidth flows, e.g., each with a packet rate of only 1 packet/second for a total of 50 packets/second throughput, this will not be possible using the scheme of [11] unless the same signing key and authentication tree data structure is shared across different flows or an average unacceptable delay of 5 seconds is imposed on each flow. Sharing the signing key and authentication tree data structure across several different unrelated flows would result in a complex software architecture at the sender end, put unreasonable restrictions on the choice of authentication mechanisms across different flows and expose privacy issues with regard to information being shared across different flows. The multiple, low-bitrate multicast flow scenario is quite likely to occur in practice since already there are sites on the internet which provide live audio and video and data feeds(such as live stock trading activity) from several different local and national sources. Currently this is done over unicast but these applications can benefit tremendously by moving to multicast. Such applications will then face the multiplicity of flow problem not just because they serve information on several different topics from several different sources but also because, in the future, handheld and embedded devices become more commonplace and information servers will need to transform each in-coming live information feed into several different output flows to cater to the different form factors and capabilities of different classes of such devices. The third practical problem with the scheme is the fact that the size of the authentication information added on to each message is not fixed but depends of the short-term packet rate which in many applications is likely highly irregular. During bursty periods, the packets will be larger and this can cause additional undesirable side-effects such as increased packet loss due to fragmentation, precisely at times when the traffic volume is large. The fourth problem with the scheme it provides no mechanism to smooth out bursty processor loads. In any real system there will be periods when the processor has enough free time to calculate several signatures/second and there will be times when the processor barely has time to calculate 1-2. With the tree approach, there is no way to leverage the idle time of the CPU to help during the time when it is highly loaded and performance will seriously degrade when the CPU gets loaded.

1.3 Introduction to our solution

We now introduce our approach which could be a possible solution to the problem of packet authentication for multicast in many scenarios and in particular does not suffer from the drawbacks described earlier. It is also likely to find applications in several other settings. Our approach provides the security guarantees required for authentication, yet is efficient in both size and speed and also works in the fully unreliable setting (with additional small but significant overhead). To do so, our approach seeks to mimic the public key signature per packet approach which would have be the best solution to this problem if the speed of well-known public key signature algorithms was not an issue.

To mitigate the problem of speed, we use as a starting point the off-line/on-line approach of [6]. The off-line/on-line approach of [6] was motivated by a need to substantially reduce the latency associated with computing regular digital signatures and also to protect regular public key signature schemes from chosen message attacks. It is well known that regular public key digital signatures schemes are slow but a single signature key, could (in practice) be used to sign an arbitrary number of messages. On the other hand, there exist very fast public key digital signature schemes based on one-way functions but, in these schemes, a single private

key could only be used to securely sign a single message or a small finite number of messages [9]. The off-line/on-line approach of [6] was the first to effectively combine these two different types of signature schemes to produce a hybrid signature scheme with several very nice properties. In the off-line/on-line approach, off-line computation is used to create buffers of 1-time key-pairs and to certify the public 1-time keys using the regular digital signature scheme. When a message needs to be signed then an on-line computation is performed to compute a signature of the message using a 1-time private key from the buffer of keys, and the corresponding 1-time public key and its certificate is also attached to this signature. Since operations on 1-time keys are extremely fast, there is very little load or latency introduced to compute message signatures. We begin by observing that since generation of 1-time key-pairs is typically also a fast operation, the off-line/on-line technique can be used to create a hybrid signature scheme which can sustain a high signature rate indefinitely even when both the off-line as well and on-line computation is being done at the same time by the same machine in parallel. The trick is to have the "off-line" computation create and certify k-time key-pairs instead of 1-time key pairs, so that the cost of the most expensive operation, i.e., certificate creation using a regular signature scheme can be amortized over k-signatures. With this technique it is easy to see that for large values of k, the sustainable rate will approach the potential signature rate possible from the k-time key-pair generation processes and for very small values of k, the rate will be close to k times the regular signature rate. By choosing an appropriate value of k, one can bring the speed of this hybrid signature scheme to be the same order of magnitude as the the speed of the k-time key generation process. Using this technique, in practice, speeds of $500-1000$ signatures/second are easily achievable for workstation class machines.

With this scheme, the multicast sender has a process or thread which generates k-time key-pairs and creates certificates for the k-time public keys using the sender's long term regular signature key. The sender uses each such k-time key pair to sign k successive messages. Depending on th reliability of the network, the certificate for the corresponding k-time public key could either sent multiple times separately or added on to each data packet or to a large fraction of the data packets. Each packet itself is signed using some i'th use of a k-time signature scheme.

It is easy to see that this basic approach solves the problems associated with unreliability, multiple flows, bursty traffic, and irregular processor loading and therefore is a good starting point towards the development of a signature scheme for this problem. Unreliability can be fully addressed by sending the k-time public key certificate in each packet or more practically addressed by sending k-time key certificates multiple times within a flow, to minimize the impact of a few lost packets. The off-line nature of the expensive signature operation when combined with very high throughput of the on-line k-time signature scheme, yields a fairly clean way to handle multiple flows, bursty traffic and bursty processor load : For each flow, buffers of k-time keys and certificates can be precomputed and filled up during periods of low CPU usage and slow traffic to tide over periods of high CPU usage and high traffic.

The major drawback of k-time/1-time signature schemes that has kept them from widespread use is that k-time/1-time public keys and/or signatures tend to be very large and thus impractical in many settings. For example, the hybrid approach outlined in the above paragraph when used in conjunction with a reasonably fast and secure k-time signature scheme, could easily result in the size overhead of the order of a kilobyte per packet, which is clearly impractical.

The main goal of this paper is to show that, by a combination of techniques, some of them quite novel and others well-known, it is indeed possible to substantially reduce the size overhead associated with the hybrid approach to make it much more practical without compromising on its security. In adopting these techniques, speed of the underlying schemes is also increased but since size is the main focus here, wherever possible we trade the speed improvements for additional size reduction.

We address the issue of size in three ways. We first focus on the size overheads in the 1-time/k-time schemes themselves and use a novel approach based on commitments and the use of Target Collision Resistant hash functions to reduce the overhead. Next we use a known but rarely used technique to reduce the size of ancillary authentication information that needs to be carried along with the signature, to identify and authenticate the i'th use of a k-time key in cases where the k-time key is created from k independent 1-time keys. Next we describe how the size of a certificate can be reduced using known techniques. These techniques allow us to create reasonably secure hybrid signature schemes which have a per-packet overhead of less than 300 bytes per packet and yet are capable of computing close to 500-1000 signatures/second on workstation class machines.

The rest of the paper is organized as follows. In section 2 we outline the ideas behind our size reduction techniques. In section 3 we provide a concrete construction of the hybrid scheme based on well known cryptographic primitives, which can be proven to be as secure as the primitives themselves. This construction also illustrates the advantages of the size reduction techniques in practice. In section 4 we analyze the performance of one version of our concrete construction if one uses functions derived from the SHA-1 compress function as the basic cryptographic primitives. In section 5 we describe some preliminary performance results of an implementation of our scheme on a PC. In section 6 we examine other applications of our signature scheme.

2 Size Reduction Techniques

2.1 TCR functions and Commitments

Most 1-time or k-time signature schemes have a signature size (and speed) which is proportional to the number of bits in the quantity being signed. Typically, this quantity is taken to be a collision resistant hash of the message being signed. In [2] it was argued that for the purposes of a signature, a weaker condition on the hash function called *Target Collision Resistance* or TCR would suffice (such functions were earlier known as Universal One-Way Hash Functions [10]). In this scenario, instead of requiring a single collision resistant hash function (which may be fairly difficult to design and which does not lend itself to a complexity theoretic analysis), a family of keyed hash functions with weaker properties could be used. The key specifies a hash function from the family. The signer of a message (the message may even be chosen by an adversary), chooses a hash function at random from the family (by picking a random key) and then computes the hash value and signs both the hash value and the key. The property of the family of hash functions, i.e., Target Collision Resistance, is that even when an ad-

versary chooses a message m_1, when the signer picks a hash function at random, with overwhelming probability (based on the choice of key) it is computationally difficult for the adversary to come up with another message m_2 such that m_1 and m_2 collide on the hash function chosen by the signer. Thus, target collision resistant hash function families may be much easier to design and have smaller output than collision resistant hash function since many attacks on collision resistant hash functions such as birthday attacks are not applicable. With this weaker notion of Target Collision Resistance, one can reduce the size of the hash of the message that needs to be signed. However in our application, this is not sufficient to reduce the size of the data which needs to be signed using the 1-time (or k-time) signature scheme since the random key used to choose a specific TCR hash function from the family needs to be signed as well. Ideally, we would have liked the sender to apply the 1-time signature only to the result of the TCR hash and not on the key. But from the definition of TCR functions, for security reasons, the signer must choose this key after the message has been fixed and this key needs to be authenticated as well. To solve this problem we adopt the following approach: Suppose that in the regular signature on the k-time public key, we include commitments to the keys used to select the TCR hash function in each of the k uses of the key. Then for each message, we could sign only its TCR hash value with the one of the k-time signatures and also reveal at the same time the key already committed to. With this approach we retain the security of the TCR construction and reduce the size of our message signatures by almost a factor of 2 and increase the speed of the underlying k-time scheme by a similar factor. Another well known fact about 1 or k-time signatures is that usually there is a tradeoff between the time it takes for key generation/signature/verification and the size of the keys and the signatures. We can trade in the factor of 2 improvement in speed we obtained by using commitments to realize additional size savings. With this tradeoff, one can, in practice, reduce the space overhead of k-time signature schemes overall by a factor of 3 to make them compact yet fast and secure.

2.2 Optimizing Ancillary Information

The next optimization we make is to the ancillary information that needs to be carried along within a k-time signature which identifies the signature as an i'th use, in situations where the k-time scheme is essentially composed of k independent one-time schemes. Typically a k-time public key is created from k one-time public keys by means of a collision resistant hash tree construction over the k public keys, with the root of the hash tree being the k-time public key. When the i'th 1-time key is used to sign a message, then apart from the 1-time signature, the i'th signature must include the i'th public key and hashes of the sibling nodes on the path from the i'th public key to the root of the hash tree. In our construction, we replace the collision resistant hash tree with a TCR hash tree [2]. This has several advantages. Firstly, the use of a TCR function implies that the sizes of all interior nodes in the tree get reduced by half and a single TCR key is used for each level in the tree. Since these level keys will be signed in the certificate for the k-time public key, they need not be sent with each signature. Thus the size overhead for the path gets reduced by a factor of 2. Also, due to reduced sizes of the intermediate nodes, a TCR tree works better with certificate signature schemes which permit message recovery from the signature, since more top level nodes can be accommodated within the embedded signed message.

However, the use of a TCR hash tree over Public keys which contain commitments transforms these commitments to weaker primitives which we term "TCR-commitments". These are no longer commitments in the true sense. We coin the term "TCR-commitment" to denote a type of commitment in which it is possible for a committer to later on alter the committed value but has no incentive to do so. However, the receiver gets no information from the commitment and also cannot create a different consistent opening of the commitment once the committer has opened the commitment. However as far as the properties of unforgeability and non-repudiation are concerned, weakening of commitments to TCR keys within individual 1-time public keys to TCR-commitments in the TCR hash tree calculation does not affect the security of underlying signature scheme.

2.3 Optimizing Certificate Size

The next optimization is to use padding schemes described in [1], to embed the k-time public key inside the regular signature within the certificate itself for many popular regular public key signature schemes, thus save on the overhead of transmitting both the k-time public key and its certificate. As mentioned earlier, this when used in conjunction with a TCR hash tree results in additional optimization since more top level TCR tree nodes can be embedded within the signature than top level collision resistant hash tree nodes.

3 Construction of our scheme from basic cryptographic primitives

We now describe our scheme which creates a n^2-time key pairs with embedded TCR-commitments to be used in a hybrid scheme involving some other regular signature scheme such as RSA. Currently, a level of security of the order of 2^{80} is considered adequate and our scheme is meant to have a comparable level of security. By adjusting the primitives described below, the scheme can be generalized to give higher levels of security.

3.1 The Primitives

Our scheme is based on the following primitives, each offering around 80 bits of security.

- Let H be a collision resistant hash function producing a 160-bit output.
- Let C be a commitment scheme for 80 bit strings. C provides a $C - create$ function for creating a commitment, a $C - decommit$ function to create a de-commitment string and a $C - open$ function for the receiver to open the commitment from the commitment and de-commitment strings. In addition for the purposes of proofs we will also assume the C is a trapdoor commitment scheme [3]. Although, at the theoretical, abstract and provable level we will use C in our scheme, at the practical construction level we will create our own commitment scheme by requiring more properties from H as described below.
- Further assume that

- H is at least a weak extractor, i.e., when H is given a high entropy input (with entropy of say least a 100 bytes), its 160-bit output is close to random. This property is fully expected to be met or even exceeded (in terms of entropy required from input) by a function like SHA-1, given its use for pseudorandom number generation.
- H is a collision resistant hash function formed by applying the MD method to a collision resistant H-Compress(k, D) function based on fixing some IV for k, where k is a key of size at least 80 bits and D is a fixed sized data block of size more than than 160 bits.
- H-Compress behaves like a family of keyed pseudorandom functions with respect the key k. This property together with the collision resistant property of H-Compress is needed to use H as part of a commitment scheme. In practice, one would realize such a function by defining H-Compress as SHA-1-Compress (and H as SHA-1) and already there are real-world applications, such as the protocol to blind credit card numbers in SET [8] which depend precisely on these two properties of SHA-1-Compress. The other difference between this commitment and one required in the theoretical case is the lack of a trapdoor. The trapdoor is an artifact of the proof technique and in practice we only need to ensure that the commitment scheme is independent of the scheme used for the TCR hash function.

With such a function, if we define $PadH(x)$ to be some fixed bijective padding of input string x upto a multiple of the block size of the compression function then $H'(r,T) = H(PadH(r)|T)$ for a large, almost random r of size at least 100 bytes, is a commitment to any string T of size less than the data block size of H-Compress, as well as a collision resistant function on r and T. The de-commit string is just r, T.

- Let $G(K,x)$ denote a family of keyed one-way functions with an 80 bit key K and 80 bit input x producing a 80 bit output.
- Let $T1(K,x)$ denote a target collision resistant keyed hash function family taking 80 bit key K and $80n$ bit input x, producing 80 bit output.
- Let $T2(K,x)$ denote a target collision resistant keyed hash function family taking an 80 bit key K and 160 bit input x producing a 80 bit output.

Before proceeding further, it will be useful to fix some notation.

NOTATION: Let f be a function from t-bit strings to t-bit strings for some t. We will use the notation $f^k(x)$, to denote k successive applications of f to the string x, e.g., $f^4(x) = f(f(f(f(x))))$. Also for any f, $f^0(x) \doteq x$.

3.2 Key Generation

The n^2-time public key is essentially derived from n^2 independent random 1-time public keys hashed down using TCR hash functions in a tree construction (see [2]). Note that by applying the concept of target collision resistance instead of collision resistant hash functions we can realize additional savings. The output of target collision resistant hash functions can be much smaller than that of collision resistant hash functions since the birthday attack is inapplicable This smaller output translates in a smaller depth of a tree based TCR hash function, since interior nodes can accommodate more child nodes.

First pick three 80-bit keys g_1, g_2, g_3 uniformly at random. These keys will be used throughout the construction of this n^2 use public key.

Throughout the construction, $g(x) = G(g_1, x)$ will be used as an 80 bit to 80 bit one-way function.

For each i, corresponding to the i'th key of the n^2-time signature scheme do the following:

Pick T_i, an 80 bit key uniformly at random. T_i will be used for selecting a TCR function when the i'th key is used to sign a message (the n^2-time public key will include a "TCR-commitment" to this key)

Pick 23 random 80 bit values, $r_{i,1}, \ldots, r_{i,23}$. From these compute $f_{i,1}, \ldots, f_{i,23}$ as follows:

For each j in $\{1, \ldots, 22\}$

$$f_{i,j} = g^{15}(r_{i,j})$$

and

$$f_{i,23} = g(r_{i,23})$$

The basic idea behind this construction (which is similar to a construction given in [6]) is that since g is a one-way function any adversary who knows $f_{i,j}$ but not $r_{i,j}$ cannot compute $r_{i,j}$ or even $g^k(r_{i,j})$ for any $k < 15$ and $1 \leq j \leq 22$.[1]

The i'th public key includes in it a "commitment" for the TCR function key to be used for signing the i'th message. Without the commitment, the public key would just be $f_{i,1}, \ldots, f_{i,23}$ or a collision resistant hash of these quantities, i.e., $H(f_{i,1}|\ldots|f_{i,23})$. To include a commitment, in a theoretical setting the public key can be defined as follows: Compute $T_i^c = C - create(T_i)$. Compute

$$PK_i = H(f_{i,1}|....|f_{i,23}|T_i^c)$$

where "|" denotes concatenation.

Instead, in practice however, we will use

$$PK_i = H'(f_{i,1}|....|f_{i,23}, T_i)$$

as both a commitment to T_i and a collision resistant hash.

The public key for the n^2-time signature scheme is then

$$PK = PK_1|PK_2|...|PK_{n^2}$$

For the sake of space efficiency (as we mentioned earlier), we do not sign a collision resistant hash of PK. Instead we sign a tree-based TCR hash of PK (as described in [2])[2]. This works as follows:

At the leaves are the n^2 public keys $PK_1, \ldots, PK_{n^2}$ each of which is 160 bits long. At the next higher level in the tree are n^2 nodes, $L_1, \ldots, L_{n^2}$, one for each public key. The value

[1] In this construction it would have been better if g was a permutation instead of a random function. The use of a one-way function reduces the entropy of $f_{i,j}$ by a few bits, i.e., even if g was a random function, 3 – 4 bits of entropy are lost after 15 iterations.

[2] Signing a TCR hash of PK instead of a collision resistant hash converts the commitments to the TCR keys T_i for message signing into a "TCR-commitment" to those keys

of node L_i is just $T2(g_2, PK_i)$, i.e., the TCR hash of PK_i, using the family $T2$ under key g_2. At the next higher level in the tree are n nodes $M_1, \ldots, M_n$. Each node M_i has n children, $L_{n(i-1)+1}, \ldots, L_{ni}$ and its value is computed as

$$M_i = T1(g_3, L_{n(i-1)+1} \mid \ldots \mid L_{ni})$$

These n nodes $M_1, \ldots, M_n$ together with g_1, g_2, g_3, constitute the public key for the n^2-time signature scheme. Each M_i has an 80 bit value, therefore the n^2-time signature scheme has a public key of size $80n + 240$ bits.

3.3 Signature Generation

To sign a message m with the i'th key of the n^2-time signature scheme, the signer does the following:

1. Compute $h = T2(T_i, H(m))$. This is an 80 bit TCR hash of m, under the assumption that H is collision resistant.
2. Let $h_1, \ldots, h_{80}$ denote the bits of h. Group these bits into 20 groups of 4 consecutive bits (aka nibbles). The value of each is a number from 0 to 15. Let $n_1, \ldots, n_{20}$ denote these numbers.
3. For $1 \leq j \leq 20$, let $Y_j = g^{15-n_j}(r_{i,j})$.
4. Let
$$N = \sum_{j=1}^{20} n_j$$
N is a number between 0 and 300 and therefore fits in 9 bits. Let n_{21} denote the value of the least significant 4 bits of N, n_{22} denote the value of the next 4 significant bits and n_{23} denote the value of the most significant 9'th bit.
5. Let $Y_{21} = g^{n_{21}}(r_{i,21})$.
6. Let $Y_{22} = g^{n_{22}}(r_{i,22})$.
7. Let $Y_{23} = g^{n_{23}}(r_{i,23})$.

The signature then consists of $T_i, Y_1, \ldots, Y_{23}$ together with the values of the $n-1$ other L nodes which corresponding to the children of $M_{(i+n-1)/n}$, excluding L_i, i.e., the siblings of L_i. Let $Sib(L_i)$ denote these nodes. Therefore the signature consists of $T_i, Y_1, \ldots, Y_{23}, Sib(L_i)$. In the theoretical setting, the signature would include T_i^c and $T_i^d = C - decommit(T_i, T_i^c)$ instead of T_i.

It has been pointed out by an anonymous referee that most of the information contained within $Sib(L_i)$ is the same among n signatures. Thus there is room for further improvement in the size overhead of a signature.

3.4 Signature Verification

To verify a signature $S_i = (T_i, Y_1, \ldots, Y_{23}, Sib(L_i))$ (or $S_i = (T_i^c, T_i^d, Y_1, \ldots, Y_{23}, Sib(L_i))$ in the theoretical setting) on a message m do the following.

1. In the theoretical setting first use $C - open(T_i^c, T_i^d)$ to obtain T_i. In the practical setting this step is omitted since T_i is provided. Compute $h = T2(T_i, H(m))$. This is an 80 bit TCR hash of m.
2. Let $h_1 \ldots, h_{80}$ denote the bits of h. Group these bits into 20 groups of 4 consecutive bits. Each group is a number from 0 to 15. Let $n_1, \ldots, n_{20}$ denote these numbers. For $1 \leq j \leq 20$, let $f_j = g^{n_j}(Y_j)$.
3. Let
$$N = \sum_{j=1}^{20} n_j$$
. N is a number between 0 and 300 and therefore fits in 9 bits. Let n_{21} denote the value of the least significant 4 bits of N, n_{22} denote the value of the next 4 significant bits and n_{23} denote the value of the most significant 9'th bit.
4. Let $f_{21} = g^{15-n_{21}}(Y_{21})$.
5. Let $f_{22} = g^{15-n_{22}}(Y_{22})$.
6. Let $f_{23} = g^{1-n_{23}}(Y_{23})$.

In the theoretical setting compute

$$PK_i' = H(f_{i,1}|....|f_{i,23}|T_i^c)$$

in the practical setting compute PK_i' as

$$PK_i' = H'(f_1|....|f_{23}, T_i).$$

PK_i' is supposed to be the same as PK_i in the keygeneration phase. Compute $L_i' = T2(g_2, PK_i')$. L_i' is supposed to be the same as L_i in the keygeneration phase. To verify this, since all the siblings of L_i are provided as part of the signature, verify that

$M_{(i+n-1)/n} = T1(g_3, < Sib(L_i), L_i' >)$

where $< Sib(L_i), L_i' >$ is just the concatenation of the L_i's in the proper order.

3.5 Proof of Correctness

In the theoretical setting, one can prove that breaking the scheme outlined above translates to breaking one of the standard cryptographic primitives used in the construction and in that sense the scheme can be considered secure. In the practical setting a similar result holds i.e., breaking the scheme means that an assumption about a primitive used is incorrect or there some unusual interaction between primitives. However, the assumptions made about primitives in the practical setting (although same assumptions have made in other deployed protocols) are much more onerous and thus more likely to be false.

4 Performance and Overhead Analysis

Analyzing the performance of the above scheme requires fixing of the primitives being used and the parameters. One needs specific functions for H, $G(K, x)$, $T1(K, x)$ and $T2(K, x)$. H can be taken to be $SHA-1$. In practice, G, $T1$ and $T2$ can all be derived from $SHA-1$ Compress and in our analysis we will assume that each application of G, $T1$ and $T2$ is comparable to an application of SHA-1-Compress.

4.1 Space Overhead

For typical values of n, (say 6), the size of the 36-time public key which is $80*6+240 = 720$ bits, is small enough so that it may be embedded inside a signature block itself for popular signature algorithms like RSA and Rabin (e.g., see [1]) which is typically expected to be at 1024 bit block size. Thus the public key and the regular signature on it will occupy 128 bytes in this case. These 128 bytes can either be broadcast frequently or attached to each packet.

The actual signature on a message m corresponding to the i'th use of the 36-time public key will have the following overhead:

- 10 bytes : Disclosure of T_i.
- 230 bytes : One time signature on $h = T2(T_i, m)$.
- 50 bytes : Values of the 5 other L nodes corresponding to the other children of $M_{\lfloor (i+5)/6 \rfloor}$, i.e., siblings of L_i.

This gives a total of 290 bytes.

Further optimization of 10 fewer bytes may be obtained in exchange for only a marginal reduction in security if the scheme is modified so that $f_{i,23}$ also serves the dual purpose of T_i.

For $n = 5$, this is reduced to 280 bytes which can be further reduced to 270 bytes by the technique described above.

An anonymous referee has pointed out scope for further space improvement by not having to repeatedly send out the almost same information on the siblings in the 6 signatures corresponding to the same M_i

4.2 Time Overhead

Clearly, the sender does one regular private key operation for the n^2-time key and the receiver does one signature verification for the n^2-time key. This time varies over the signature algorithm and can be amortized over the n^2 uses. We now focus on the overhead just for the n^2 use key.

Key Generation

For efficiency it may be better for the signer to store all intermediate results in the key generation process. For n^2 uses of the signature, the total amount of overhead is:

1. $240n^2 + 30$ random byte generation. In practice this is likely to be through a pseudorandom process.
2. $22 \times 15n^2 + n^2 = 331n^2$ applications of g.
3. n^2 applications of H on 240 bytes.
4. n^2 applications of T2.
5. n applications of T1.

To get a sense of how many operations these are if we assume for example that an application of $T2$, $T1$, g is comparable to a $SHA-1$ compress, one application of H on 240 bytes of data is comparable to 5 $SHA-1$ compress and one $SHA-1$ compress produces 20 random bytes, this time overhead is comparable to $349n^2 + n + 2$ $SHA-1$ compresses, or on an average 349 $SHA-1$ compresses per key. On high end workstations, 1000 1-time keys can be produced per second.

Signature Generation

If tables are kept at the time of key generation, then the cost of the signature is essentially computing the TCR hash of the message and around 25 table lookups.

Signature Verification

Signature verification is very similar to key generation. However, on an average only 8 applications of g would be needed instead of 15 for each of the nibbles in the TCR hash of the message and no random numbers need to be generated. Thus total number of operations are

1. $8 \times 22 + 1 = 177$ applications of g.
2. 1 application of H on 240 bytes.
3. 1 application of $T2$
4. 1 application of $T1$.

Carrying the $SHA-1$ analogy, this translates to roughly 184 applications of $SHA-1$ Compress.

5 Implementation Experience

We have implemented a version of this scheme as part of an ongoing prototyping effort to develop an architecture for secure multicast [4]. The implementation was done in "C" on a 400Mhz Pentium II machine running Linux. We implemented a 36-time hybrid scheme with SHA-1 as the main component together with 1024-bit RSA as the long term signature algorithm. We used the pthreads threading package on Linux to create a separate thread for the off-line processing for each signature context. This thread is responsible for replenishing a key buffer of upto 1800 uses. The preliminary results are as follows: If 1024-bit RSA directly used to sign packets then the maximum packet rate achievable was found to be approximately 40 packets/s (for RSA, we recoded parts in assembly). With the hybrid solution, a rate of around 410 packets/s could be sustained. For small packets, the latency of the on-line signature calculation (provided that a 36-time key was already available) was around $70\mu s$, provided that the signature calculation thread did not get pre-empted by the key-generation thread. To simulate irregular and bursty packet rate we ran an application which would randomly and uniformly select the number of packets to send per second in the range of 100 to 600 and found that current system and buffer levels were able to handle this load with ease. The numbers are also likely to improve once some of the other time critical routines (such as SHA-1) are re-coded in assembly.

6 Other applications

Our scheme which is essentially a very fast and reasonably compact approach to creating digital signatures from basic cryptographic primitives should enable a large number of applications which require a high signature rate. In scenarios, where the size of the signature does not matter, we can trade back the size advantages we have obtained in our scheme to get much faster hybrid 1-time/k-time signature schemes than were known before. In applications, where size of the signature matters, we are likely to have one of the best signature rates for a given size.

One other practical situation where our technique can find good use is in the export-controlled ciphersuite implementations within the SSL protocol. One of the peculiarities of SSL is that when an export-controlled client connects to an SSL server, the handshake consumes more time at the server end than when a full-crypto client connects, even though the security of the key agreement has been reduced from 1024-bit RSA to 512-bit RSA! One would normally expect this modulus size reduction to result in the export-controlled connection being 8 times faster on the server side. The reason for this discrepancy is the following: For export controlled connections, in the SSL protocol the server has a 1024-bit certified signature key and it generates ephemeral 512-bit encryption keys for key agreement. Typically these 512-bit keys are used for around 100-500 connections. Now, upon receiving a client hello, the server uses its certified 1024-bit key and the received client random nonce to create a certificate for the current 512-bit ephemeral encryption key. The client then uses the 512-bit key for key agreement. Therefore a connection with an export-controlled client costs the server one 1024-bit signature and one 512-bit decryption and is therefore more onerous than a more secure connection involving only 1024-bit encryption. Since there is no reason why the server should spend more resources to provide less security, we can use our scheme to reduce the server load when connecting with export-controlled clients to be commensurate with the security provided, i.e., reduce the load back to 512-bit decryption. For this we can employ our hybrid scheme based on generating 100/500-use keys (depending on the server policy on the lifetime of the ephemeral 512-bit encryption key) and create a single certificate for them using the server's 1024-bit signature key. For each new connection request for which an ephemeral 512-bit encryption key needs to be used, we can use the 100/500-use scheme to generate a signature on the current 512-bit key and client nonce and return that to the client.

7 Acknowledgments

The author would like to thank Hugo Krawczyk for pointing out the benefits of using Target Collision Resistance for this application, Shai Halevi, for invaluable help in the design of the scheme and to Shai Halevi and Charanjit Jutla for help in the design of the primitives. The author would also like to thank Ran Canetti and Rosario Gennaro for helpful discussions. This paper has also benefitted from comments received from anonymous referees.

References

[1] M. Bellare, P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. *in Advances in Cryptology - EUROCRYPT '96*, Lecture Notes in Computer Science 1070, Springer-Verlag, 1996.

[2] M. Bellare, P. Rogaway. Collision-Resistant Hashing: Towards Making UOWHFs Practical. *in Advances in Cryptology - CRYPTO '97*, Lecture Notes in Computer Science 1294, Springer-Verlag, 1997, pp 470–484.

[3] G. Brassard, D. Chaum and C. Crepeau. Minimum disclosure proofs of knowledge. *JCSS.* 37(2):156–189, 1988.

[4] R. Canetti, P. Cheng, D. Pendarakis, J. Rao, P. Rohatgi and D. Saha. An Architecture for Secure Internet Multicast, Internet Draft, draft-irtf-smug-sec-mcast-arch-00.txt, Feb 25 1999, Work In Progress.

[5] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. *IEEE INFOCOM '99.*

[6] S. Even, O. Goldreich, S. Micali. On-Line/Off-Line Digital Signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

[7] R. Gennaro, P. Rohatgi. How to sign digital streams. *in Advances in Cryptology - CRYPTO '97*, Lecture Notes in Computer Science 1294, Springer-Verlag, 1997, pp 180–197.

[8] H. Krawczyk. Blinding of credit card numbers in the SET protocol.*Financial Cryptography '99.*

[9] L. Lamport. Constructing Digital Signatures from a One-Way Function. *Technical Report SRI Intl.*, CSL 98, 1979.

[10] M. Naor, M. Yung. Universal one-way hash functions and their cryptographic applications. *Proceedings of the 21st Annual Symposium on Theory of Computing*, 1989.

[11] C. Wong, S. Lam. Digital Signatures for Flows and Multicasts. *Proceedings IEEE ICNP '98*, Austin TX, Oct 1998.

[12] R. Merkle. A Certified Digital Signature. *Advances in Cryptology – CRYPTO '89*, 1989.