SANDIA REPORT

SAND2003-4105
Unlimited Release
Printed November 2003

Hybrid Cryptography Key Management

Cheryl Beaver, Michael Collins, Timothy Draelos,
Donald Gallup, William Neumann, and Mark Torgerson

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2003-4105
Unlimited Release
Printed November 2003

Hybrid Cryptography Key Management

Cheryl Beaver, Michael Collins, Timothy Draelos,
Donald Gallup, William Neumann, and Mark Torgerson
Cryptography and Information Systems Surety Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0785
{cbeaver, mjcolli, tjdrael, drgallu, wneuman, mdtorge}@sandia.gov

Abstract

Wireless communication networks are highly resource-constrained; thus many security protocols which work in other settings may not be efficient enough for use in wireless environments. This report considers a variety of cryptographic techniques which enable secure, authenticated communication when resources such as processor speed, battery power, memory, and bandwidth are tightly limited.

Contents

Figures

Tables

Hybrid Cryptography Key Management

1 Introduction

In general, wireless communication networks are resource constrained. These constraints include bandwidth, battery power, processing power, and memory among others. As technology advances, processing power and memory constraints will lessen; however, it is likely that there will always be a difference between the resources of a wireless and a wired communication device. Security and communication protocols developed for a wired network may or may not be feasible for use in a wireless network. The resources available to the wireless network may not be sufficient to facilitate the wired protocols. Further, there are fundamental differences in the physical communication media that simply do not allow a direct translation of protocols that sit at the lowest levels of the communication stack.

This report looks at a broad collection of issues and ideas associated with the resource constrained nature of wireless networks. In particular, we focus on cryptographic protocols and ideas that will lead to more efficient security for wireless networks. Even though the work was conducted with the resource constraints of wireless networks in mind, much of what is presented also applies to wired networks.

To date the security community has a very large assortment of basic cryptographic primitives to draw from. For the most part, proper application of these primitives may satisfy the security needs of most wired networks. Because of the large body of cryptographic primitives in existence there are not a lot of obvious avenues to pursue when attempting to create a system applicable to resource constrained environments. We have chosen two paths to examine. The first is to consider networks of a restricted type in order to develop primitives that are tailored to the needs of the network. The second is to pursue less known technologies to determine their applicability to the needs at hand.

1.1 Network Models

We consider several particular types of networks which arise frequently in applications. Some of the key-management solutions considered in this report are specific to only one type of network. We can minimize resource use by taking advantage of

the properties of particular network types, and by not providing services that are unnecessary for a particular type.

- **Star Network:** This is a network in which there is a central hub which can communicate with all other nodes (of which there may be many), and there are no direct communication links between non-hub (or "peripheral") nodes (Figure 1). If two peripheral nodes need to communicate with one another, all messages must be routed through the hub. This kind of topology arises naturally in many military scenarios, in which there is a central command center controlling a collection of subordinate units. Frequently the hub node has far more computational power than the peripheral nodes.

- **Hierarchical Network:** This may be seen as a generalization of a star network. Communication links have the structure of a rooted tree, with a single node at the root (Figure 2). The root can communicate with a small number of subordinate nodes; each subordinate communicates with a small *disjoint* set of sub-subordinates and so on until "leaves" or "end-user" nodes are reached. Each end-user has a unique path of communication up to the root.

- **Ad-hoc Network:** This is a network in which there is no pre-established topology or hierarchy. Individual nodes dynamically enter and leave the network. Nodes must discover one another's existence and work out adequate communications paths. The lack of structure makes such networks difficult to manage; but in many situations it might be too expensive, or simply impossible, to determine ahead of time who will be in the network and how they will be connected. Note that in an ad-hoc situation there may be an external "trusted party" that facilitates communication (by acting as a certificate authority for example), but the trusted party does not control how users communicate.

2 Key Management Schemes

The development of key management schemes arises from the need to manage keys for network communications using symmetric key algorithms, while satisfying the following requirements:

- *Join or Past Secrecy* - New members cannot read past messages of the group.

- *Leave or Forward Secrecy* - Leaving members cannot read future group messages.

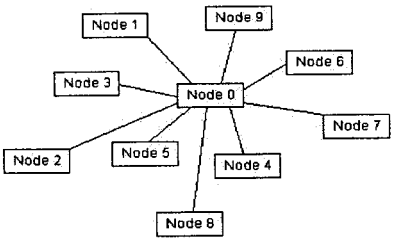

Figure 1. Star Network

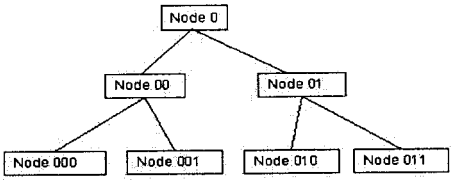

Figure 2. Hierarchical Network

These requirements imply that the cryptographic network key used by group members must be changed both when a new member is added to the group and when a member

leaves the group. Multiple key management schemes continue to emerge because of the need to minimize bandwidth, storage, and computational costs subject to these constraints. Key management schemes also assume a communication ability to transmit messages to more than one group member at the same time. In general there are three kinds of communications in a network:

- **Broadcast:** A message is sent which can be heard by everyone in the network. An example would be a radio signal which can be received by everyone within range of the transmitter.
- **Unicast:** A message is sent to one recipient. This is the norm for an IP packet, which has a single destination address.
- **Multicast:** A message is sent to a selected group of recipients. Internet protocols have been developed which enable a sender to send a message to a single IP address and have it delivered to multiple recipients [1]; this is more efficient than sending the same message multiple times. In some cases we can achieve multicast by broadcasting one message which has been encrypted in such a way that all the intended recipients, and only they, can decrypt it.

Below, we present a few schemes for managing network keys among group members (see [8] for a comparative presentation of key management schemes in multicast communication environments). The number of users in the network is denoted by N.

2.1 Group Key Management Protocol (GKMP)

One of the simplest key management protocols is called the Group Key Management Protocol (GKMP) and it supports a network based on the Star architecture. In this architecture, there is a single key manager (KM) for all members of the communications network. Each user stores 2 keys: the *group key* held by all members, and a unique key-encryption-key (KEK) held only by a single member and the KM. The KM holds the KEK for every member plus the group key for a total of $N + 1$ keys. A user's unique key is called a KEK because it is used to securely transmit the group key.

When a member leaves the group, the group key must be updated for the remaining group members in order to maintain forward secrecy of network communications. For GKMP, the leave operation requires $N-1$ unicast transmissions of the new group key.

When a member joins the group, the group key must be updated for the existing group members in order to maintain past secrecy of network communications. For GKMP, the join operation requires one multicast transmission of the new group key to existing members plus a unicast transmission of the new group key to the new member.

The GKMP is very simple, but extremely difficult to manage in large, constantly changing groups because of the difficulty of updating the group key when a member leaves.

2.2 Scalable Multicast Key Distribution (SMKD)

The Scalable Multicast Key Distribution (SKMD) protocol is an attempt to address the problems of scaling to large groups that are inherent with the GKMP. The underlying topology of SMKD is a hierarchical network; however it has some aspects of an ad-hoc network since the tree is built up dynamically. Initially there is just the root node, which authorizes other nodes to act as key managers for their own groups; these nodes may in turn authorize other key managers and so on until the end-users are reached. SMKD makes use of internet multicast protocols [1] to build this hierarchy. Although better than GKMP, SKMD does not necessarily scale well to large groups since there is no limit on the size of a given group.

2.3 Complementary Key Scheme (CKS)

The Complementary Key Scheme (CKS) is a key management approach designed to minimize the cost of a leave operation at the expense of key storage space. It supports the star architecture with a single KM for a group of N network members. In addition to its own KEK and the group key, each member stores $N - 1$ "complimentary variables", one for each of the *other* group members. In order to implement a leave operation, the KM broadcasts a single cleartext message containing the index of the leaving member. The new group key will be some known deterministic function of the old group key and the complimentary variable of the leaving member. Thus a leave operation is very inexpensive, but as the group becomes large, the storage costs become prohibitive.

2.4 DISEC

The Distributed Framework for Scalable Secure Many-to-Many Communication (DISEC) key management scheme distributes key management tasks among the group members, thus avoiding dependency upon a single KM. DISEC uses a virtual binary tree, where group members are the leaves of the tree. "Virtual" means that the internal nodes of the tree are mathematical abstractions, not actual machines or users. Each member generates their own secret key, k, and computes a hash of this secret key, bk, which it shares with its sibling. An internal node's secret key is a function of hashed child keys, $k_{parent} = m(bk_{child0}, bk_{child1})$. Each user must know the blinded keys of the siblings of the nodes on its path to the root. Given this information, the user can compute all the secret keys along its path to the root. The root key is thus a function of all the member's hashed keys and is used as the group key. Detailed analysis of join and leave operations can be found in [9, 19].

2.5 Hierarchical Tree Structure (HTS)

The most scalable of the key management approaches presented here uses a hierarchical tree structure (HTS) to organize keys. The HTS approach uses a d-ary key management tree, where group members are the leaves of the tree. Each user stores $h = \log_d N + 1$ keys, the keys on the one path from the leaf to the root of the tree. Since every user will store the root of the tree, it is used as the group key and the leaf keys, being unique to each leaf, are KEKs. The KM stores the entire tree, for a total of $(dN - 1)/(d - 1)$ keys. In the case of a binary tree, the KM stores $2N - 1$ keys. This approach has acceptable scalability attributes since the number of keys stored by the user scales logarithmically in N, as do the join and leave operations, which we analyze in section 2.5.1.

2.5.1 HTS Analysis

In this section, we take a closer look at the HTS approach to key management, given that it is the best of the presented schemes with respect to scalable group key management in dynamic multicast environments. There are three primary ways to deliver keys to group members even under the same HTS key management architecture.

- **User-oriented rekeying** - In user-oriented rekeying, a key delivery message contains the exact information (keys) needed by a specific user or group of users.

The KM asks the question of each group member, "What keys do you need?" and delivers those set of keys to that member. One can see that this approach may not take advantage of multicast communication.

- **Key-oriented rekeying** - In key-oriented rekeying, a key delivery message contains a single key and it is delivered to everyone who needs it. The KM asks the question for each key, "Who needs this key?" and delivers it to those members who need it. One can see how this can potentially require more messages than necessary if users share some keys.

- **Group-oriented rekeying** - In group-oriented rekeying, a key delivery message contains as many keys as a particular group or subgroup of members needs. The KM asks the question, "What keys does this group need as a whole?" and delivers the set of keys to the group.

The key-oriented rekeying approach results in the most, but shortest messages and the group-oriented approach results in the fewest, but longest messages. One way of measuring the cost of operations within the HTS key management approach is to measure

1. total number of messages sent by the KM,
2. the size of the group receiving the message, and
3. the size of the message sent.

Table 1 shows comparative costs associated with a complete rekey of the group from a key-oriented and user-oriented perspective. From the table, it is clear that key-oriented rekeying uses more, smaller messages than user-oriented rekeying.

Another way of measuring the cost of HTS operations is to measure the number of encryptions/decryptions required by the KM or a group member [33]. Any time a key is communicated, it must be encrypted/decrypted using a key held by the sender and all intended recipients. Table 2 presents average encryption/decryption costs of rekeying on different elements of the network using the key-oriented approach.

From Table 2, we can derive the average KM cost of an operation as (recall that $h = \log_d N + 1$):

$$\text{Average KM Cost} = (d+2)(h-1)/2 = (d+2)(log_d N)/2. \quad (1)$$

	Key-oriented			User-oriented		
	Messages	**Group Size**	**Message Size**	**Messages**	**Group Size**	**Message Size**
	N	1	k	N	1	$(n+1)k$
	N/d	d	k			
	N/d^2	d^2	k			
	$\vdots$	$\vdots$	$\vdots$			
	d^2	N/d^2	k			
	d	N/d	k			
	1	N	k			
Total	$\frac{d}{d-1}N$		$\frac{d}{d-1}kN$	N		$(n+1)kN$

Table 1. The cost of rekeying the entire group under key-oriented and user-oriented rekeying, where $n = log_d N$ and k = key size.

	Requesting Member	**Non-requesting Member**	**Key Manager**
En(De)cryptions for Join	$h-1$	$d/(d-1)$	$2(d-1)$
En(De)cryptions for Leave	0	$d/(d-1)$	$d(h-1)$

Table 2. Average encryption/decryption costs of rekeying different elements of a network using key-oriented rekeying with a d-ary tree and N users; $h = \log_d N + 1$.

A plot of the average KM cost versus the degree, d, of the KM tree for $N = 1000$ is provided in Figure 3, where one can see that the cost is minimized for $d = 4$.

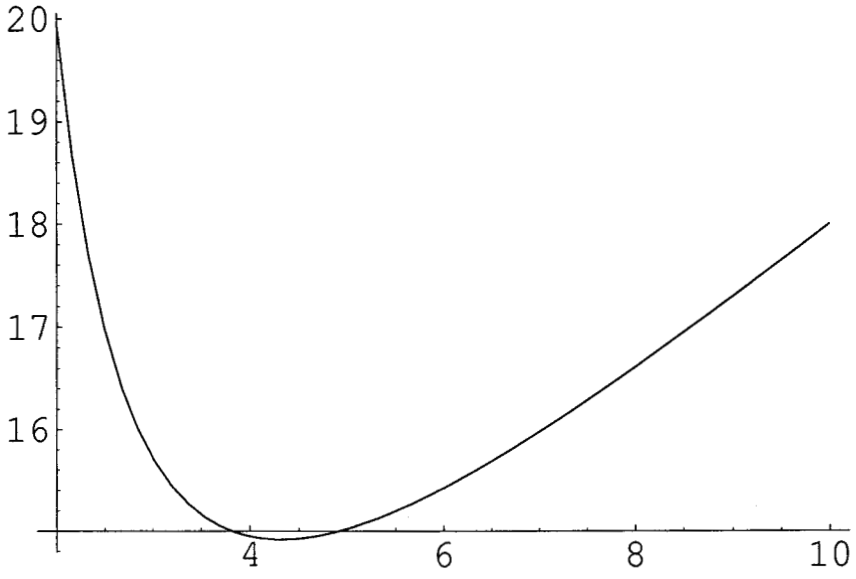

Figure 3. Plot of the average number encryption/decryptions for leave/join operations for the KM as a function of the degree of the tree for $N = 1000$.

3 Special Solutions

Here we describe a variety of schemes that address fundamental cryptographic problems and which are tailored for specific types of resource constrained environments.

3.1 On-line/Off-line signatures

Digital signatures can take several milliseconds and require considerable power to compute. This can be a problem in time-critical situations, when there are a very large number of messages to sign in a short time (high throughput), Sor for low-power devices. One solution to this problem is an on-line/off-line signature scheme. The idea is to save time during the on-line phase of a digital signature by performing the time-consuming part of the computation off-line and storing the result for later use. When a message needs to be signed, a fast computation is done on-line to finish the signature [11].

A disadvantage of on-line/off-line schemes is that message lengths are considerably

longer than with usual signature schemes (the following scheme, due to Shamir and Tauman [29], approximately doubles the size of the signature). Also, this scheme assumes that we have available both considerable storage space (in which to store the off-line portion of the signatures) and considerable idle time (in which to do the precomputation).

The scheme makes use of "trapdoor hash functions". A trapdoor hash function is a special type of hash function which is collision resistant unless the user has knowledge of a special trapdoor key. More specifically, given a hash function $H(m, r)$, it is in general computationally infeasible to find $m' \neq m, r'$ such that $H(m, r) = H(m'r')$, but given a trapdoor key, TK, such a pair m', r' can be found easily (polynomial time). In particular, suppose m is a message and r is a random number. Given the trapdoor key, TK, and a second message, m', there is a polynomial time algorithm to find a second random number r' such that $H(m, r) = H(m', r')$. Given such a trapdoor hash function, H, an on-line/off-line signature scheme works as follows:

- Off-line: Precompute a set of random (m, r), $H(m, r)$ and signatures on $H(m, r)$ (using any signature algorithm).
- On-line: Given a message, m' to sign, choose one of the precomputed signatures on a random message (m, r) and compute r' such that $H(m, r) = H(m', r')$ using the trapdoor key, TK. Transmit r' and the precomputed signature on $H(m, r)$.

The time for the on-line portion of the signature depends on how long it takes to find r'. The authors in [29] refer to this as a *hash-sign-switch* scheme for obvious reasons. Their trapdoor hash function is constructed as follows. Choose at random two *safe* primes, p, q (i.e., such that $p' = \frac{p-1}{2}$ and $q' = \frac{q-1}{2}$ are also prime) of length $k/2$. Set $n = pq$. Let g be a random element of order $2p'q'$ in $\{\mathbb{Z}/n\{\mathbb{Z}$. Then $H(m, r) = g^{m|r} \pmod{n}$ where $m|r$ denotes the concatenation of m and r. The trapdoor to this function is the factorization of n: p, q. Given a second message, m', and the trapdoor p, q to find a collision, one need only solve the equation $2^k m + r \equiv 2^k m' + r' \pmod{2p'q'}$, i.e., $r' = 2^k(m - m') + r \pmod{2p'q'}$. Since r' needs to be sent with the signature, the size of the signature is approximately doubled.

The on-line portion of this signature scheme is very fast: the authors in [29] estimate that finding r' takes one-tenth as much time as performing a single modular multiplication on 1024-bit numbers. This is assuming that n is a 1024-bit number and the messages m, m' are 160 bits; then r, r' are about 1024 bits. In comparison, a typical RSA signature requires modular exponentiation of a 1024-bit number. The scheme does have some drawbacks that may or may not be considered significant

depending on the application. In particular, since r' needs to be sent along with the signature, the bandwidth required for the signature is increased. Furthermore, if a lot of signatures are required, then either lots of storage space is necessary or there is danger of running out of pre-computed signatures. The latter problem could be mitigated by computing more signatures during system idle time if there is sufficient power available.

3.2 Public/Symmetric Hybrid Key Distribution Scheme.

One common issue faced in a network of users is distributing keys to subgroups of users. The goal is to minimize the number of keys generated, maintained or distributed by/to each user, while maximizing the number of subsets with distinct secret keys. We assume that there exists a trusted third party to manage the keys, as in a star network; without one, this problem becomes very cumbersome.

There are many proposed schemes to solve this problem. The idea presented here is still in development, and in its current form does not lead to a gain in efficiency, but we believe is a promising start and an interesting new approach to the problem. Our design uses both symmetric and public key techniques. We would like a scheme to publicly distribute group keys in such a way that only valid group members could deduce the secret group key. In our scheme, we assume there is a Trusted Key Generating Authority (TKGA) that shares a unique secret with each member of the group of m users $(s_1, s_2, \ldots, s_m)$. When a group key for users $(u_1, u_2, \ldots, u_n)$ is needed, the TKGA uses the s_i to generate a function similar to a trapdoor function in the sense that any of the users who know any one of the secrets s_i can get a common output, k, from the function, but any user who does not know one of the s_i's cannot. The value k will then be the common key.

Example: The TKGA shares a pair (s_i, p_i) with each user i. Here p_i is a large prime number and s_i is an integer modulo p_i. To distribute the key k to a group of users $U = \{u_1, .., u_n\}$, the TKGA does the following:

1. Let $x_i = (k \oplus s_i) + y_i * p_i$ for each $i \in U$ (where y_i is some random integer and we treat $k \oplus s_i$ as an integer as well so x_i is an integer).

2. Set $F(x) = \sum_{i \in U} x_i * \left(\frac{\prod_{j \neq i}(x - s_j)}{\prod_{j \neq i}(s_i - s_j)} \right)$

Then to compute k, user i computes $k = s_i \oplus (F(s_i) \pmod{p_i})$. Thus k is easy to compute for any user who knows some s_i used in the construction of F (e.g. the intended group members), but hard for anyone else. This means F does not have to be protected (much like public keys) and hence makes the distribution problem easier.

We believe this is an interesting idea, but the solution example given above does not really offer great improvement over existing schemes. For example, the amount of data needed to describe the public function, F, is as great as the amount of data that would be needed to encrypt k with a different secret key for each user. It may save in transmission costs since F can be posted in some public location instead of having to transmit a separate message to all n users. We hope to improve this idea by finding a function F that has a simpler description. One idea is to allow some information leakage to some of the other users who are non-group members (but not to non users of the system). This may be acceptable if the leakage is minimal or only to a subgroup of users, especially if there is some degree of trust in the users to whom the information is leaked.

3.3 Identity-based schemes

Cryptographic schemes based on the mathematics of elliptic curves have proven to be a secure way to reduce key size, computation and bandwidth requirements on key exchange and digital signatures. Any discrete log based cryptographic algorithm can be converted to use elliptic curve arithmetic. Attacks on the usual modular arithmetic version of a discrete log problem are not generally effective when elliptic curve arithmetic is substituted. Hence, elliptic curve based discrete log systems may use smaller key sizes and as a result will generally have lower computation requirements and shorter signatures.

Much effort has gone into devising methods to exploit the attractive features of elliptic curve methodologies. For instance, Sandia has developed a highly optimized elliptic curve digital signature hardware chip [28]. Other directions of research have to do with so called identity-based schemes, for which elliptic curves are ideally suited. The most promising make use of bilinear pairings on the curve. Identity based schemes exist for key exchange, encryption and digital signature. See [4], [13], [6], [14] for more details.

The fundamental idea of an identity based scheme is that a user's name can be that user's public key. In general, a public key is a long, hard-to-remember number.

In this system, a user's public key might be `jane_doe@provider.com` – more precisely, a combination of such a name with some publicly available information. Hence a user of the system, who knows the naming convention for public keys, would be able to know the public key of a user without needing to consult with a trusted third party (TTP). The user, Jane Doe, would get the private key corresponding to the public key `jane_doe@provider.com` from a TTP that generates the keys for all users of the system. The TTP also publishes the aforementioned public information needed to transform arbitrary names into public keys.

Another component of a PKI is a certificate revocation list. The purpose of a CRL is to maintain a list of certificates that have been revoked (for example, if keys are lost or stolen). Users of a PKI periodically consult a CRL to make sure that public keys have not been compromised. Identity-based systems can design in this feature to make CRLs less crucial. To do this, a temporal element is included in the public key. For example `jane_doe_Feb272003@provider.com` would be the public key for Jane Doe good on February 27, 2003 only. Jane would have to get a new key from the TTP daily. This limits damage if a key is lost or stolen. Without checking a CRL, a user knows that a certain key was valid at some point in the very recent past. But to guarantee that a key is valid right now, a CRL would still be needed.

Another useful feature is that messages could be encrypted for later decryption; use the key `jane_doe_Dec252003@provider.com` to encrypt a message that couldn't be read until Dec. 25.

An identity based system requires interaction with a trusted third party, but does enable communication between parties who a priori don't know whether the other exists, but know what the identity of a trusted party should be if it was there and in the position to communicate. Hence this could be effective in ad-hoc networks or in any system where a stable infrastructure may not be readily available or desirable.

The major disadvantage of identity-based schemes is that far more trust must be given to a central authority. In a standard PKI, it is possible to devise the system in such a way that the certificate authority (CA) merely verifies the link between an entity and a public key, and does not know the private keys of the users. In all known identity-based schemes, the TTP generates and hence knows all the private keys. This means that if the TTP does not play fairly it may decrypt all messages in the system. Another drawback is that non-repudiation in the strictest sense is lost. We have non-repudiation only to the extent that the TTP is trusted, but not in any cryptographic sense. In commercial applications, these weaknesses may render such schemes inappropriate. On the other hand, in military applications or high consequence commercial systems where end users may not be given the ability to

generate their own keys, full trust in the TTP is required.

3.4 Joint Authentication and Encryption

In an environment constrained by speed and latency issues, even traditional block ciphers may be too slow. Typical block ciphers follow the Feistel structure and may require a large number of rounds (e.g. 10 or more) for adequate security. For example, DES is quickly broken by linear or differential cryptanalysis [7] if the number of rounds is reduced much below the specified sixteen. Several alternatives have been suggested to create small round Feistel ciphers (e.g. 3-5 rounds). The reference [16] gives a number of attacks that may or may not be applicable to low round constructions depending on the exact cipher description. In a separate report [?], we also have studied these issues. In particular, we consider the problem of combining encryption and authentication into a single, fast algorithm. The result is a cipher mode that has many of the same properties as CBC mode of encryption. However the mode can be parallelized and also contains an authentication step that is essentially free.

3.5 Password-Based Systems

As multi-user computer systems became more prevalent, a need was realized for secure authentication protocols to restrict access to the various servers only to authorized users. At the time, a number of authentication protocols were known, however, they required the user to memorize a random string that could be hundreds or thousands of bits long. As a result, new protocols were developed that only required the user to memorize a short, usually text based password. Unfortunately these protocols suffered from a number of security flaws, including susceptibility to dictionary attacks and sending the users password to the server in the clear, so it could easily be "sniffed" off of the network. In a separate report [23], we survey a number of more recent password based authentication protocols that solve this authentication problem in a secure and efficient manner.

3.6 HORSE

Source authentication of messages is a valuable tool in communication networks where the source of a message can easily be spoofed. Often, when two parties are communicating, a shared-key message authentication code is sufficient for providing source

authentication to both parties. Unfortunately, the utility of a MAC does not extend to a broadcast situation involving three or more parties, so public-key digital signatures have traditionally been used to attain source authentication in group broadcast communications. In a separate report [22] we introduce HORSE, an r-time signature scheme that yields source authentication in the group setting like a public-key signature scheme, only with signature and verification times much closer to those of a MAC. Additionally, HORSE makes much more efficient use of its keys than previous r-time signature schemes.

The advantages of r-time signature schemes are very fast signing and verification; these operations require just a few hash function evaluations. The disadvantage is that such a scheme requires a long key which can be used for only a few messages. This r-times-only property turns out to be sufficient for some applications; see [24, 25].

3.7 The FX Construction

Let F be a block cipher with block length n and key length m. Encryption with key k is denoted by F_k. Rivest proposed a construction for increasing the security of F with very little additional computation. The block cipher FX is defined by

$$FX_{k_x k k_y}(p) = k_y \oplo F_k(p \oplo k_x)$$

Obviously $k_x, k_y \in \{0,1\}^n$, giving a key length of $2n + m$. The use of exclusive-ors before and after encryption is called prewhitening and postwhitening; k_x and k_y are called "whitening keys" to distinguish them from the "central" key k.

This is an appealing construction from the standpoint of computational efficiency: encryption and decryption with FX take hardly any more time than with F. This is in sharp contrast to triple encryption, which is the usual method for getting around the inadequate key length of DES. Furthermore it would be straightforward to extend a hardware implementation of F to FX. In spite of its simplicity, this construction appears to be quite effective, giving a considerable increase in the effective key-length. Thus FX is useful for low-power situations, or any situation in which speed is an issue. It can also extend the life of legacy hardware or software with inadequate encryption. One disadvantage is key length; we need to add two bits to the overall key in order to gain at most one additional bit of security. This is because there is no need to search on k_x and k_y separately; any possible value of kk_x uniquely determines k_y since $k_y = c \oplus F_k(p \oplus k_x)$ (where p, c are plaintext and ciphertext).

3.7.1 Slide Attack Against FX with Known Central Key

One plausible way of using the FX construction, which might be appealing in a wireless or other tightly resource-constrained environment, is to let the central key k be a network key shared by all members of a group; then any two members of the group cold communicate using a private (k_x, k_y) pair. This would be of particular interest when F is a cipher such as Blowfish [27], for which the overhead of setting up the key-schedule and key-dependent s-boxes is considerable. In this case k can have a relatively long lifetime, while k_x and k_y are easy to change and could be changed frequently. The best attack known against this construction is the "slide attack" of [3]. The idea of this attack is to look for two plaintext/ciphertext pairs $(p, c), (p^*, c^*)$ such that $c \oplus c^* = k_y$. Note that for such a pair we have $c^* = F_k(p \oplus k_x)$; so one of the intermediate steps of the process of encrypting p gives an output equal to the final result of encrypting p^*. Relationships of this kind, equating the ith stage of one encryption or decryption with the jth stage of another, are the basis of slide attacks.

How can we find such a pair? For any such pair we would have

$$p = k_x \oplus F_k^{-1}(c \oplus k_y) = k_x \oplus F_k^{-1}(c^*)$$

and

$$p^* = k_x \oplus F_k^{-1}(c^* \oplus k_y) = k_x \oplus F_k^{-1}(c)$$

thus

$$k_x = p \oplus F_k^{-1}(c^*) = p^* \oplus F_k^{-1}(c)$$

and

$$p^* \oplus F_k^{-1}(c^*) = p \oplus F_k^{-1}(c). \tag{2}$$

Now we can think of $p \oplus F_k^{-1}(c)$ as a hash of (p, c); using well-known collision-finding techniques [20], we expect to find $(p, c), (p^*, c^*)$ satisfying (2) after examining about $2^{n/2}$ p/c pairs. We have just seen that (2) must hold whenever $c \oplus c^* = k_y$; this happens with probability 2^{-n} for any randomly chosen c, c^*. If $c \oplus c^* \neq k_y$, then the two sides of (2) are uncorrelated random strings and will be equal with probability 2^{-n}. So if (c, c^*) is chosen uniformly at random from the set of all pairs of ciphertexts satisfying (2), the probability of $c \oplus c^* = k_y$ is given by Bayes' rule:

$$\frac{2^{-n}}{2^{-n} + (1 - 2^{-n})2^{-n}} \approx \frac{1}{2}.$$

Therefore half the collisions we find will satisfy $c \oplus c^* = k_y$. There is no difficulty in identifying these; let $k'_x = p \oplus F_k^{-1}(c^*)$ and $k'_y = c \oplus c^*$, then see if encrypting $\hat{p}$ with

$k'_x k k'_y$ produces $\hat{c}$ for some pair $(\hat{p}, \hat{c})$ different from (p, c) and (p^*, c^*). We expect to find a valid collision given $2^{(n+1)/2}$ known p/c pairs. Note that this attack requires only known (not chosen) plaintexts.

Thus we conclude that, with k already known, the keys k_x, k_y provide at most $(n+1)/2$ bits of security if the lifetime of k is long enough to permit an attacker to collect $2^{(n+1)/2)}$ p/c pairs. The level of security might be greater when the lifetime of k is shorter. When k is unknown, the attacker can repeat the slide attack for each possible value of k. This may be impractical, but it is a considerable improvement over exhaustive search on $k_x k k_y$. This repeated slide attack comes close to the theoretical bounds of [10] and [15]. Those papers analyze "pure" key-search attacks on the FX construction, i.e., attacks which treat F as a black box or a random permutation. If it is possible to attack F itself by linear or differential cryptanalysis, then these techniques can extend to better attacks on FX. For a detailed tutorial on linear and differential cryptanalysis, with applications to the FX construction, see [7].

4 Experimental Testbed

The development of an experimental testbed is important for the evaluation of candidate key management protocols. The emphasis here is to gain empirical evidence about the communication, computation, and storage costs of various key management protocols. We have developed a network benchmarking tool and a key distribution simulator using the HTS key management scheme (section 2.5); we have also implemented several important cryptographic primitives, developed a general software architecture that allows incorporation of new algorithms, and implemented a graphical user interface.

The Java programming language was chosen as the development language for the testbed. Java comes with source code of most of what we have in our crypto library and much more source code is available from the public domain. We do have some optimizations in our library that are not in the Java package, but they can be added if necessary. The advantages of Java are portability, cleaner code without explicit pointers, strong object-orientation, and easier development of graphical user interfaces and networking. Furthermore, Java appears to be on a high-growth curve with respect to its use in military and other applications. The advantage of C is speed; but our performance comparisons, described in section 5.3, indicate that the difference in speed is not too large.

4.1 Key Distribution Simulator

The key distribution simulator allows a user to describe a hypothetical network and to model the communications required for distributing keys in such a network. The emphasis is on determining the amount of computation and communication needed to set up and maintain such networks.

There are seven sets of variables that can be selected to set up the simulator: 1) network type and size, 2) the message transmission approach used by the network, 3) the encryption approach (and key size) for distributing the keys, 4) physical terrain type over which the network is distributed 5) node types for each node, 6) processing capability for each node, and 7) the transmission frequency and range for each node. The simulator was initially intended to model communications in wireless networks. As a result, a number of the variables listed above were included as input but were not fully implemented. For example, physical terrain types such as mountainous regions that could impact wireless transmission are not implemented. However, there are "hooks" in the code (in both the input sections and the calculational sections) that would allow all of these variables to be implemented with minimal effort.

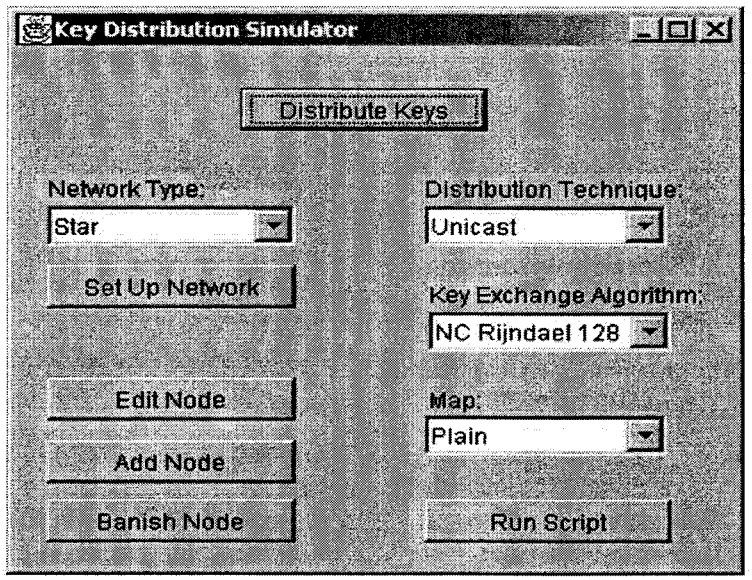

Figure 4. The Main Dialog Box

The first step for the user in setting up a simulation is to set the distribution technique, key exchange algorithm, and map to be used. This is done using the three drop-down lists on the right side of the main dialog box (Figure 4). The two

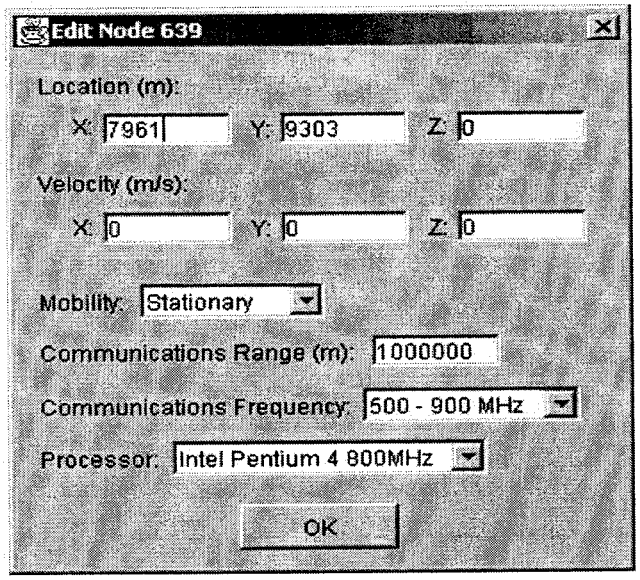

Figure 5. Edit Node Dialog Box

distribution techniques available are "Unicast" and "Broadcast." (Note: The only map available is for a "Plain." Any other selection from this list will default to "Plain." The Plain is a flat grassland that is 10,000 meters (10 km) on a side.)

Next, the user sets up the network. Currently, the star network (Figure 1) is the only functioning option. The user clicks on the "Set Up Network" button; this causes a dialog box to appear that accepts the number of nodes to be placed in the network. When the "OK" button on this box is pushed, the network is set up. As a part of this setup, the individual nodes are created and placed randomly on the map.

After the network has been set up, nodes can be added, banished (i.e., removed) and edited. A node is added by pushing the "Add Node" button on the main dialog box. A node can be banished by pushing the "Banish Node" button on the main dialog box. This causes a second dialog box to appear; the user enters the number of the node to banish in this box. To edit the properties of an individual node, the user pushes the "Edit Node" button on the main dialog box. This causes a new dialog box to appear; the user enters the number of the node to edit into this dialog box. Upon pushing the "OK" button, the "Edit Node" dialog box appears (Figure 5). This box can be used to edit a node's location, velocity, mobility, communications range, communications frequency, and processor type (the impacts of communications frequency are not currently implemented). Any changes to the nodes properties take

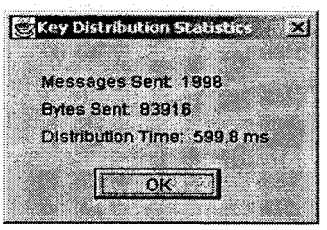

Figure 6. Key Distribution Statistics Dialog Box

```
# Network Type
# 1 - Star, 2 - Wagon Wheel, 3 - Hubless Wagon Wheel
0
# Number of Nodes
15
# Terrain Map
# 0 - Plain, 2 - Forest, 3 - Mountains
0
# Key Exchange Algorithm
# 0 - NC Rijndael 128, 1 - NC Rijndael 256, 2 - Algorithm3
0
# Distribution Technique
# 0 - Unicast, 1 - Multicast, 2 - Broadcast
0
# Node Data
#  X       Y       Z       Vx     Vy     Vz   Mobility  CommRange  CommFreq  CPU
00001.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00002.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00003.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00004.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00005.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00006.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00007.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00008.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00009.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00010.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00011.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00012.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00013.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00014.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
00015.0 00000.0 00000.0  000.0  000.0  000.0     0       1000000       0      0
# Actions
distributekeys
```

Figure 7. Example Script File

effect when the "OK" button is pressed.

Once the network is set up, an estimate the work required by a network to dis-

tribute cryptographic keys to the nodes on the network is obtained by pressing the "Distribute Keys" button on the main dialog box. This creates a dialog box with the results of the calculation, using the HTS key management scheme to distribute keys. An example of the "Key Distribution Statistics" dialog box is shown in Figure 6.

A second method for obtaining an estimate of the work required to distribute keys across a network is to run a script file that contains all of the information about the network. An example script file is shown in Figure 7. To run a script file, the user presses the "Run Script" button in the lower right hand corner of the main dialog box. This creates a dialog box that requests the name of the script file. The user enters the name of the script file and presses the "OK" button. The "Key Distribution Statistics" dialog box appears with the results of the calculation. Using script files allows the user to create variations on a network and determine the impact of these variations on key distribution.

4.2 HTS Implementation

The HTS key management approach can be implemented with a tree degree of 2 as a balanced binary tree. The following characteristics of a balanced binary tree ease the implementation:

- All leaf nodes are at level h or $h-1$.
- All leaves at level h are justified left in the tree.
- The tree node at index i has children at $2i$ and $2i+1$ and its parent at $i/2$.

One valuable aspect of using a balanced binary tree is that the tree can be implemented using a one-dimensional array, where the indexing is determined according to the third bullet above.

One issue that arises in any tree-oriented data structure is balancing the tree when leaves are removed during a leave operation. Initially, an approach was used of "filling" the hole created in the tree by a member leaving the group. This was done to maintain the same structure of the tree after a leave operation and was relatively easy to implement. The hole was filled by the last leaf in the tree, the tree was restructured, and keys were updated as needed. This technique works just fine if used only once or in very limited ways. However, the potential exists for a node to be moved around the tree during multiple leave operations such that if that member

ever leaves, it will be extremely difficult to determine which keys in the tree need updating.

Subsequently, a new approach of keeping track of holes in the tree is currently used in the implementation. During a join operation, holes are filled prior to adding new nodes to the tree. The following functions of the HTS multicast key management scheme are currently implemented with this approach:

- **Key Distribution** - This operation rekeys the entire group of members and populates the KM tree using key-oriented rekeying.
- **Key Update** - This operation rekeys a single group member, updating all the necessary keys in the KM tree using key-oriented rekeying.
- **Leave (Remove) Operation** - This operation removes a group member, updates the necessary keys to preserve forward secrecy of the group, and keeps track of the hole left by the leaving member.
- **Join (Add) Operation** - This operation adds a member to the group and updates the necessary keys to preserve past secrecy of group communications.

5 Implementation Performance

This section discusses three implementation issues. The first is the implementation of a benchmark program to compare the relative performance of various cryptographic algorithms, both encryption and authentication, on the same hardware platform. The second is the implementation and performance of Mark Torgerson's enhanced exponentiation algorithm (EEA) and how it compares with other enhancement techniques. The third issue is the relative performance of the Java and C programming languages on operations important to cryptography. All tests were performed on the following hardware.

- Dell Precision 340 Desktop computer
- 2.53 GHz Pentium IV Processor
- 512 KB Processor Cache
- 512 MB RDRAM
- 133 MHz Bus Speed

5.1 Cryptographic Benchmark Tests

Software benchmarking allows comparative performance evaluations of cryptographic algorithms. Performance of algorithms can be measured with respect to speed and memory usage (both program memory and data memory). However, precise memory usage figures can be difficult to acquire in an automated manner because the code used to measure memory usage changes the measurement. In addition, the code necessary to perform a particular operation is often dispersed among many functions in multiple files. Simply totaling the size of the object code in an application does not give an accurate memory usage figure because much of the code may not ever be used by a certain operation being measured. In contrast, speed can be measured in a non-invasive manner and test code can be easily wrapped around specific functions of interest, even if lower-level calls are made to functions in different files.

Different compilers provide different optimizations. In particular, in-line assembly instructions for rotating registers is available in the Microsoft Visual C++ 6.0 compiler, but not in BorlandC++ 6.0. Without a rotate instruction, one must use two shift instructions, one left and one right, and logically OR the result. Therefore, at a minimum, the rotate operation can be executed three times faster with a rotate instruction than without. The rotate operation is necessary in both the SHA-1 and MD5 hash algorithms as well as in the Advanced Encryption Standard (AES) [12].

The benchmark program utilized the following software environment and was executed on the previously mentioned hardware. All algorithms were tested with 10,000 byte messages.

Software specifics:

- Microsoft Visual C++ 6.0 Service Pack 5 with Processor Pack
- Crypto++ Library 5.1 [30]
- 32-bit in-line assembly rotate instructions

Table 5.1 presents speed measurements of selected encryption algorithms. The algorithm expected to be the most secure is also the fastest. The TEA algorithm [31] is known for its simplicity and, therefore, its small code size.

Table 5.1 presents speed measurements of selected hash algorithms and authentication algorithms. The HMAC algorithm [17] uses a hash algorithm to provide message authentication with minimal addition cost.

Table 5.1 presents speed measurements of selected algorithms for authenticated encryption. MTC4, RMTC4, and GCSA are algorithms that use information from the internal state of the cipher to provide the authentication [2]. The encryption has properties similar to CBC mode, yet the encipherment and authentication mechanisms can be parallelized and/or pipelined. The authentication overhead is minimal, so the computational cost of the authenticated encryption is very nearly that of the encryption process. Also, the authentication process remains resistant against some IV reuse. OCB [26] is another parallelizable authenticated encryption algorithm that operates faster than performing encryption followed by HMAC for authentication.

Algorithm	**Speed (Mbytes/sec)**
AES-128	88
TEA	31
DES	33
3-DES	13

Table 3. Results of benchmark tests on selected encryption algorithms.

Algorithm	**Speed (Mbytes/sec)**
MD5	201
SHA-1	57
HMAC-MD5	199
HMAC-SHA-1	56
TEA Auth	21

Table 4. Results of benchmark tests on selected hash, authentication, and digital signature algorithms.

5.2 Enhanced Exponentiation Algorithm

The enhanced exponentiation algorithm replaces the pseudo-random number generator (PRNG) and exponentiation functions required by DSA signature generation. It

Algorithm	Speed (Mbytes/sec)
OCB-AES-128	71
MTC4-SHA-1	10
MTC4-MD5	22
RMTC4-SHA-1	15
RMTC4-MD5	28
GCSA-AES-128-AES	66
GCSA-AES-128-SHA-1	65
GCSA-AES-128-MD5	66
AES-128-CBC-HMAC-SHA-1	33
AES-128-CBC-HMAC-MD5	58

Table 5. Results of benchmark tests on selected authenticated encryption algorithms.

requires approximately 1 kilobyte of precomputed storage space, assuming a 1024-bit base and modulus and a 160-bit exponent, and results in an PRNG/exponentiation function that is approximately 9 times faster than using Montgomery exponentiation [21] alone. The speed is comparable to the best fixed-base modular exponentiation speed-up that we have implemented thus far, the BGMW algorithm [5], at a fraction of the storage cost. Table 6 shows the performance of several enhancement approaches to modular multiple-precision integer exponentiation implemented in C and their impact on DSA signature generation.

The EEA and BGMW algorithms are both precomputation schemes, where powers of the base are computed and stored for later use in exponentiations. This reduces the number of multiplications necessary for exponentiations with random exponents. The Montgomery algorithm can be used in both EEA and BGMW for the multiplications that must be computed during an exponentiation. In Table 6, "big mps" refers to the size of multiple precision integers used for the base and modulus and "small mps" refers to the size of the exponents. The times in the table are based on computations using a 1024-bit base and modulus and a 160-bit exponent.

Exponentiation Algorithm	Signature Time (ms)	RNG + Exp Time (ms)	Storage
EEA (m=5)	5.6	4.9	6 big mps + 6 small mps
EEA (m=6)	5.2	4.4	7 big mps + 7 small mps
BGMW	8.2	7.4	80 big mps
BGMW	4.2	3.6	3400 big mps
Montgomery	41	40	none

Table 6. Performance of enhancement algorithms for modular multiple-precision integer exponentiation implemented in C.

5.3 Performance Comparison of Java and C

The Java programming language has a reputation of performing considerably slower than the C language. One reason is that it is not a compiled language, but runs interpretively on a Java Virtual Machine (JVM), which is a software program implemented for a particular microprocessor. Java is optimized at run-time, instead of compile-time like C. Other reasons are that in Java, bounds-checking is performed on all array accesses and dynamic memory allocation is handled through a general garbage collector. However, bounds checking, garbage collection, and run-time optimizations have advantages as well, not the least of which are in the area of security (e.g. prevention of buffer overflows).

We conducted performance comparisons between Java and C on basic arithmetic and cryptographic operations and found Java to be very competitive with C. Our results are presented in Table 7. In some cases we did not have identical algorithms implemented in each language, so we could not make a direct comparison. Nevertheless it is clear that well-written Java is fast enough to be a reasonable choice for cryptographic operations. The following programs were compared.

1. *Matrix multiplication* - This test consisted of identical C and Java code for matrix multiplication, which is similar to the kinds of integer operations often used in multiple-precision integer arithmetic.

2. *Modular exponentiation* - This test consisted of a C and Java implementation of modular multiple-precision exponentiation using a 1024-bit base and modulus and a 160-bit exponent. The C implementation used a Montgomery enhance-

ment [21] and the Java implementation used a sliding window of size 3 as an exponentiation enhancement (Algorithm 14.85 in [20]).

3. *DSA signature generation* - This test consisted of a C implementation of DSA signature generation using EEA for modular exponentiation and a Java implementation using a sliding window of size 3 for exponentiation.

4. *DSA signature verification* - This test consisted of a C implementation of DSA signature verification using the BGMW algorithm [5] for modular exponentiation and a Java implementation using a sliding window of size 3 for exponentiation. EEA could not be used to speed up the exponentiation because it requires a random number generation aspect, which is needed in signature generation, but not signature verification.

Function	**C (ms)**	**Java (ms)**
Matrix Multiply - Identical Code	28	40
Modular Exponentiation	40 (Montgomery)	24 (Sliding Window)
DSA Signature Generation	5 (EEA)	25 (Sliding Window)
DSA Signature Verification	9 (BGMW)	48 (Sliding Window)

Table 7. Running times of Java and C on cryptographic operations.

References

[1] A. Ballardie, "Scalable Multicast Key Distribution", Network Working Group (IETF), RFC 1949, 1996.

[2] C. Beaver, T. Draelos, R. Schroeppel, and M. Torgerson, "ManTiCore: Encryption with Joint Cipher-State Authentication", SAND Report 2003-1488C.

[3] A. Biryukov and D. Wagner, "Advanced Slide Attacks", in proceedings of Eurocrypt 2000.

[4] D. Boneh and M. Franklin, "Identity Based Encryption from the Weil Pairing", in Proceedings of Crypto 2001, LNCS 2139, pp. 213-299, 2001.

[5] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation, in Advances in Cryptology—Proceedings of Eurocrypt '92, Vol. 658, pp. 200–207, Springer-Verlag, Berlin/New York, 1992.

[6] J. Cha and J. Cheon, "An Identity Based Signature from Gap Diffie-Hellman Groups", cryptology e-print archive (eprint.iacr.org), 2002/18.

[7] M. Collins, "Cryptanalysis of the FX Construction", SAND report 2003-xxxx.

[8] S. Dahlman, "Key Management Schemes in Multicast Environments", Master's Thesis, University of Tampere, November, 2001.

[9] L. R. Doneti, S. Mukherjee, and A. Samal, DISEC: Distributed Framework for Scalable Secure Many-to-many Communication, in Proceedings of the Fifth IEEE Symposium on Computers and Communications, 2000.

[10] S. Even and Y. Mansour, "A Construction of a Cipher from a Single Pseudorandom Permutation", in J. Cryptology, v. 10 no. 3 (Summer 1997), pp. 151-162 (earlier version in Asiacrypt '91).

[11] S. Even, O. Goldreich, and S. Micali, "On-Line/Off-Line Digital Signatures", in Proceedings of Crypto '89, LNCS 0435, 263-275, 1990.

[12] Department of Commerce/NIST, "Advanced Encryption Standard," FIPSPUB 197, November 26, 2001.

[13] C. Gentry and A. Silverberg, "Hierarchical ID-Based Cryptography", in Proceedings of ASIACRYPT 2002, LNCS 2501, pp. 548-566, 2002.

[14] F. Hess, "Exponent Group Signature Schemes and Efficient Identity Based Signature Schemes based on Pairings", cryptology e-print archive (eprint.iacr.org), 2002/12.

[15] J. Kilian and P. Rogaway, "How to Protect DES Against Exhaustive Key Search (an Analysis of DESX)", in J. Cryptology v. 14 No. 1 (Winter 2001) pp. 17-35.

[16] L. Knudsen, "The Security of Feistel Ciphers with Six Rounds or Less", in Journal of Cryptology, vol. 15 no. 3 (Summer 2002), pp. 207 – 222.

[17] H. Krawczyk, M. Bellare, R. Canetti, "HMAC: Keyed hashing for message authentication," Internet RFC 2104, February 1997.

[18] H. Krawczyk and T. Rabin, "Chameleon Hashing and Signatures", in Proceedings of Network and Distributed System Security Symposium, NDSS 2000, pp. 143-154, 2000.

[19] A. McGrew, A. T. Sherman, 'Key Establishment in Large Dynamic Groups Using One-Way Function Trees", IEEE Transactions on software engineering, 1998.

[20] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, 1997.

[21] P. L. Montgomery, "Modular Multiplication Without Trial Division", Mathematics of Computation, Vol. 44, No. 170, April 1995, pp 519-521.

[22] W. Neumann, "HORSE", SAND report 2003-xxxx.

[23] W. Neumann, "EKE", SAND report 2003-xxxx.

[24] A. Perrig, "The BiBa one-time signature and broadcast authentication protocol", in Eighth ACM Conference on Computer and Communication Security", pp. 28-37, November, 2001.

[25] L. Reyzin and N. Reyzin, "Better than BiBa: Short One-time Signatures with Fast Signing and Verifying", Cryptology ePrint Archive, Report 2002/014, http://eprint.iacr.org/.

[26] P. Rogaway, M. Bellare, J. Black, and T. Krovetz, "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption", In Proc. 8th CCS, pp. 196-205, ACM, 2001.

[27] B. Schneier, "Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish)", R. Anderson, ed., Fast Software Encryption, Cambridge Security Workshop (LNCS 809), pp. 191-204, Springer-Verlag, 1994.

[28] "A Low Power Design for a Digital Signature Chip" by R. Schroeppel, C. Beaver, T. Draelos, R. Gonzales, R. Miller in proceedings of Cryptographic Hardware and Embedded Systems (CHES) 2002.

[29] A. Shamir and Y. Tauman, "Improved Online/Offline Signature Schemes", in Proceedings of Crypto 2001, J. Kilian (Ed.), LNCS 2139, pp. 355-367, 2001.

[30] W. Dai, "Crypto++ Library", http://www.eskimo.com/weidai/cryptlib.html.

[31] D. J. Wheeler and R. M. Needham, "TEA, a Tiny Encryption Algorithm", B. Preneel, ed., Fast Software Encryption (FSE), Second Annual Workshop (LNCS 1008), pp. 363-366, 1995

[32] D. Wong and A. Chan, "Efficient and Mutually Authenticated Key Exchange for Low Power Computing Devices", in Proceedings of ASIACRYPT 2001, LNCS 2248, pp. 272-289, 2001.

[33] C. Wong, M. Gouda, and S. Lam, "Secure Group Communications Using Key Graphs", IEEE/ACM Transactions on Networking, vol. 8, no. 1, Febrary 2000.

DISTRIBTUTION:

1	MS 0785	W. E. Anderson, 6514
1	0785	C. L. Beaver, 6514
1	0785	M. J. Collins, 6514
1	0785	D. R. Gallup, 6514
1	0785	A. J. Lanzone, 6514
1	0785	T. S. McDonald, 6514
1	0785	W. D. Neumann, 6514
5	0785	M. D. Torgerson, 6514
2	0899	Technical Library, 9616
1	9018	Central Technical Files, 8945-1
1	0323	D. Chavez, LDRD Office, 1011