

Efficient Key Distribution Schemes for Large Scale Mobile Computing Applications

Mahalingam Ramkumar
 Department of Computer Science and Engineering
 Mississippi State University.

Abstract

In emerging networks consisting of large-scale deployments of mobile devices, efficient security mechanisms are required to facilitate cryptographic authentication. While computation and bandwidth overheads are expensive for mobile devices, the cost of storage resources continue to fall at a rapid rate. We propose a simple novel key predistribution scheme, *key subset and symmetric certificates* (KSSC) which can take good advantage of inexpensive storage resources, and has many compelling advantages over other approaches for facilitating ad hoc establishment of pairwise secrets in mobile computing environments. We argue that a combination of KSSC with a variant of an elegant KDS proposed by Leighton and Micali is an appealing choice for securing large scale deployments of mobile devices.

I. INTRODUCTION

Rapidly lowering costs of computing and communication devices is expected to result in explosive growth of mobile computers equipped with wireless capabilities. In the “wireless future” home equipment like TVs, DVD players, refrigerators, microwaves, garage openers, security cameras etc., equipped with short range radios, will be controlled by hand-held devices like PDAs or mobile phones, even from remote locations. Sensors monitoring vital internal organ functions of a person on the road may relay early warning signs over multi-hop ad hoc networks to the nearest hospital to facilitate timely responses.

An important requirement for the feasibility of such co-operative networks is a mechanism for *cryptographic authentication* of devices deployed in *very large scales*. Cryptographic authentication is realized through *security associations* (SA) facilitated by key distribution schemes (KDS).

A. Key Distribution Schemes

Popular KDS include Kerberos-like schemes based on the Needham-Schroeder symmetric key distribution protocol [1], and the public key infrastructure (PKI) used in conjunction with asymmetric cryptographic primitives.

Kerberos-like schemes which require an on-line trusted server to mediate interactions between entities for establishing SAs are not suitable in scenarios where access to the trusted server is not possible. Thus key distribution schemes that facilitate *ad hoc* (without *active* involvement of a third party) establishment of SAs are required.

While public key schemes (in conjunction with PKI) cater for this requirement they demand substantial bandwidth and communication overheads for exchanging public key certificates and performing asymmetric computations. As deployments are expected to include resource constrained devices, we would like to restrict the devices to employing only inexpensive symmetric cryptographic primitives. At the same time however, we require such key distribution schemes to support *very large* network scales. Unfortunately schemes based only on symmetric primitives *do not scale well*. More specifically, scalable schemes either ① require a trusted server for mediation (Kerberos-like schemes), or ② are *susceptible to collusions* (key predistribution schemes).

1) *Key Predistribution Schemes*: Scalable *key pre-distribution* schemes (KPS) [2] consist of a key distribution center (KDC) who chooses a set of P secrets and a practically unlimited number (say N) of entities with unique identities. Each entity is assigned a set of $k < P$ secrets by the KDC, which are *derived* from the secrets chosen by the KDC. KPSs use only inexpensive symmetric cryptographic primitives. Any two entities (say A and B) can independently compute an SA (a common secret) K_{AB} by using the secrets assigned to them by the KDC.

While there is no practical limit on the number of entities N who can be provided with secrets, a group of *colluding* entities can *pool their secrets together* to compromise the KPS. An n -secure KPS can resist collusions of up to n entities pooling their secrets together, *irrespective* of the network size N . Any KPS is essentially a *trade-off between security and complexity*. A measure of the security is n , the number of colluding nodes that a KPS can resist. Arguably, if a KPS can resist collusions of several tens of thousands or even hundreds of thousands of entities, their “susceptibility to collusions” may not be a practical issue. However increasing n results in increased *complexity*.

The metrics for complexity can include several factors like computation, bandwidth, and storage overheads. For applications involving mobile computing devices computation and bandwidth overheads are expensive. However the cost of storage continues to reduce at a very rapid rate, with no impending sign of “Moore’s law saturation.” Flash based storage cards supporting several GBs are already common, and are very well suited for mobile devices. Thus employing a few tens of megabytes of that storage for the storing KDS parameters is indeed practical.

Increasing collusion resistance of *most* KPSs call for ① increasing the number of secrets k (and thus storage for secrets) assigned to each entity *and* 2) increased *computational complexity* for evaluation of SAs. While the increase in storage requirement is not a serious issue (after all, storing one million 64-bit secrets requires a mere 8 MB of storage), high computational complexity is *not* acceptable.

2) *Contributions*: In this paper we argue how the reducing cost of storage requirement can offset the inherent limitations of inexpensive symmetric cryptographic primitives. The specific contributions of this paper are two fold. The first is a novel, yet simple key predistribution scheme, KSSC, which (we argue) is particularly well suited for facilitating ad hoc establishment of SAs in mobile computing applications. As a second contribution we argue why i) the elegant key distribution scheme proposed by Leighton and Micali (LM-KDS) in [3] is very well suited for small-scale networks and that ii) a *combination* of LM-KDS and KSSC is an appealing choice for securing such networks.

B. Organization

In Section II we provide an overview of some KDSs and explore several sources of complexities associated with KDSs. In Section III we introduce the KSSC scheme and provide an extensive analysis of its performance and security-complexity trade-offs. In Section IV we investigate practical issues associated with key distribution. We argue why the LM-KDS [3] is very well suited for small-scale deployments and that a combination of LM-KDS and KSSC is very well suited for large scale networks. In Section V we discuss some related prior work. We discuss some similarities between KSSC and a *broadcast* authentication scheme proposed by Canetti et al [4] which also employs symmetric certificates [5] and key subsets. Few other key predistribution schemes that have been proposed in the literature are also evaluated. Conclusions are offered in Section VI. The following notations are used in this paper

- 1) $h()$ - a secure hash function (for example, SHA1);
- 2) A, B - identities of entities;
- 3) K_{AB} - a symmetric shared secret between entities A and B ;
- 4) $h(\mathcal{M}, K)$ - a hashed message authentication code (HMAC) for a message \mathcal{M} using a secret K .

II. KEY DISTRIBUTION FOR EMERGING NETWORKS

A key distribution scheme (KDS) is a mechanism for distributing secrets and / or public values to all participants (entities) to facilitate establishment of cryptographic bonds or security associations (SA)

between the entities. The most common of such SAs are one-to-one SAs which take the form of a *shared secret* between two entities.

A shared secret K_{AB} between entities A and B can be employed for privacy of exchanges between A and B , and mutual authentication. In a scenario where entity A sends a message \mathcal{M} to entity B , the message can be encrypted with the shared secret K_{AB} if *privacy* of exchanges is desired. In scenarios where *authentication* of the source and the *integrity* of the message \mathcal{M} is desired, A can append a hashed message authentication code (HMAC) $h(\mathcal{M}, K_{AB})$ to the message \mathcal{M} .

KDSs can be broadly classified into *nonscalable* and *scalable* schemes. An example of a nonscalable scheme is the basic key predistribution scheme, where to support a network of N entities a key distribution center (KDC) chooses $\binom{N}{2}$ pairwise secrets and provides each entity with $N - 1$ secrets.

An example of a scalable KDS is Kerberos-like schemes, which however require a trusted server for mediating SAs. While the basic KPS does not require involvement of the KDC after the entities are supplied with secrets, the need for $\mathcal{O}(N)$ storage for each entity restricts the number of entities that can be supported. In Kerberos-like approaches while each entity needs to store just one secret to authenticate itself to the trusted server, they do not facilitate *ad hoc* establishment of SAs.

A. Scalable KDSs

Scalable KDSs that facilitate ad hoc establishment of SAs can be categorized into certificates-based and ID-based schemes.

1) *Certificates Based Schemes*: In certificates based schemes each entity is associated with an ID, a public key, and a private key. Each entity is free to *choose* their own private key and compute the corresponding public key. An entity with ID A can choose a private key R_A and derive (compute) a public key U_A . As the public key U_A provides no information about the identity A , a trusted third party has to securely specify a *binding* between the identity and the public key of every entity. This binding is typically achieved through a *certificate* issued by a certificate authority (CA).

Well known certificates based schemes include several asymmetric encryption and signature schemes like RSA, El Gamal (and variants like DSA) and elliptic curve schemes, used in conjunction with a CA or more often a hierarchical organization of CAs in the form of a public key infrastructure (PKI). Such schemes facilitate pairwise SAs by exchanging chains of public key certificates and performing some computations.

2) *ID-Based Schemes*: For ID-based schemes the ID of an entity itself doubles as the public key, thus obviating the very need for certificates. A key distribution center (KDC) chooses public parameters of the system and one or more *master* secrets. The KDC can then compute the private key(s) corresponding to *any* public key (ID). The private keys for an entity with identity A are thus *assigned* by the KDC to the entity A .

ID-based schemes are increasingly seen as preferable over certificates based schemes for large scale networks, and especially for many emerging application scenarios like ad hoc networks. In existing networks based on the client-server paradigm, the client and server exchange public key certificates for mutual authentication, at the end of which a shared secret is established. This secret can be used for authentication and encryption of a *large* number of packets exchanged between them subsequently. Thus the overheads (exchange of certificates and their verification) incurred for establishing a shared secret can be leveraged for securing large amounts of data. However in ad hoc networks a node will typically exchange small packets with *many* nodes. Thus the overheads for exchanging certificates with each node may be prohibitive. With ID-based schemes two nodes A and B can independently compute a shared secret K_{AB} without exchanging certificates for this purpose.

In ID-based schemes the ID of an entity is typically chosen as a secure one way function of a descriptive real-life identity and credentials. For example Alice with real-life identity described by string $S_A = \text{“Alice B Cryptographer, AnyTown, USA, \dots”}$ can be assigned an ID $A = h(S_A)$. In such scenarios *large* “ID spaces” are required to ensure that collisions are rare. For a network expected to support up to 2^t

entities, the IDs should be larger than $2t$ -bits. Furthermore pre-image resistance of the hash function $h(\cdot)$ is also required to ensure that A cannot substitute the preimage S_A with an alternate one - say S'_A where $A = h(S'_A) = h(S_A)$.

ID-based public key schemes for encryption (IBE) and signatures (IBS), most of which take advantage of pairings in special elliptic curve groups, have attracted substantial attention recently [6].

B. ID-Based Key Predistribution Schemes

ID-based key predistribution schemes (KPS) consist of a KDC and entities with unique IDs drawn from a large ID space. For example, if 128-bit IDs are used the size of the ID space is $N = 2^{128}$. The KDC chooses a set of P master secrets \mathbb{S} and each entity is provided with a set of $k \leq P$ secrets. The set of k secrets \mathbb{S}_A assigned to an entity with ID A is determined by a public function $\mathcal{F}(\cdot)$ which takes two inputs - the ID A , and the set of P KDC secrets \mathbb{S} . Two entities A and B (with secrets \mathbb{S}_A and \mathbb{S}_B respectively) can discover an SA in the form of a pairwise secret K_{AB} using a public function $\mathcal{G}(\cdot)$. In other words

$$\left. \begin{array}{l} \mathbb{S}_A = \mathcal{F}(\mathbb{S}, A) \\ \mathbb{S}_B = \mathcal{F}(\mathbb{S}, B) \end{array} \right\} \text{ and } K_{AB} = \mathcal{G}(\mathbb{S}_A, B) = \mathcal{G}(\mathbb{S}_B, A) \quad (1)$$

An attacker who has access to secrets of many (say v) entities $\{O_1 \cdots O_v\}$ may be able to discover K_{AB} even *without* access to \mathbb{S}_A or \mathbb{S}_B . An n -secure KPS can resist collusions of up to n entities pooling their secrets together.

With $\mathcal{O}(n)$ limitation on storage, the basic KPS can only support a network size of $N = \mathcal{O}(n)$. However the basic KPS is *not* susceptible to collusions. On the other hand, scalable KPSs can support *any* N , but can only *tolerate* collusions of up to n entities. That such trade-offs are possible was first realized by Blom et al [7] who proposed the first KPS in the literature.

1) *Blom's SKGS*: In Blom's SKGS (symmetric key generation system) [8] based on maximum distance separation (MDS) codes over $GF(q)$ (say the finite field $\mathbb{Z}_q = \{0, 1, \dots, q-1\}$ where q is a prime), a public MDS generator matrix \mathbf{G} with primitive element α is used. For a network size of $N \leq q$ the $(n+1) \times N$ MDS generator matrix is $\mathbf{G} = [\mathbf{g}_0 \cdots \mathbf{g}_{N-1}]$ where \mathbf{g}_j s are column vectors of length $n+1$ with the i^{th} element given by $g_j(i) = \alpha^{ij}, 0 \leq j \leq N-1, 0 \leq i \leq n$.

The KDC chooses a $(n+1) \times (n+1)$ symmetric matrix \mathbf{D} with $\binom{n+1}{2}$ independent values (secrets) chosen randomly from \mathbb{Z}_q . Entity A is assigned $k = n+1$ values (secrets) of $\mathbf{d}^A = \mathbf{D}\mathbf{g}_A$. A and B (with secrets \mathbf{d}^A and \mathbf{d}^B respectively) can calculate $K_{AB} = (\mathbf{d}^A)^T \mathbf{g}_B = (\mathbf{d}^B)^T \mathbf{g}_A$ (as D is a symmetric matrix). In other words, A computes K_{AB} as

$$K_{AB} = \sum_{i=0}^n d_i^A \beta_i \text{ where } \beta_i = \alpha^{iB}. \quad (2)$$

The choice of the value $q > N$ will depend on the desired size of the ID space N . If we desire 128-bit IDs we should choose at least 128-bit q . The k secrets assigned to any entity will also be 128-bits long. Computation of a shared secret involves finite-field multiplication operations employing k secrets, and k finite-field exponentiation operations. An n -secure SKGS is unconditionally secure as long as n or less entities pool their secrets together. However, it is *completely* compromised if more than n entities do so - or the failure occurs catastrophically.

C. ID-Based Probabilistic KPSs

More generally KPSs can be regarded as (n, p) -secure, where an attacker can expose a fraction p of all possible SAs by pooling together secrets assigned to n entities. As long as p is low enough (say 2^{-64}) it is *computationally infeasible* for an attacker to even identify *which* SAs can be compromised by using the pooled secrets. Deterministic n -secure KPSs (like SKGS) can be seen as special cases of (n, p) -secure "probabilistic" KPSs where p takes only binary values (0 or 1). While deterministic KPSs

fail catastrophically ($p(n') = 0$ for $n' \leq n$ and $p(n') = 1$ for $n' > n$), for (n, p) -secure probabilistic KPSs p increases gracefully with n .

Most probabilistic KPSs are based on the idea of random (or pseudo random) allocation of subsets of keys to each entity, from a pool of keys chosen by the KDC. Such probabilistic KPSs are extensions of similar techniques [9] - [10] that relied on deterministic strategies for allocation of subsets of keys to every node. Dyer et al [11] were the first to point out that “set intersection schemes are best generated at random.” Approaches based on random subset allocation have since been used for broadcast authentication [4], and for securing interactions between sensors in ad hoc sensor networks [12] - [15].

1) *Random Preloaded Subsets*: Schemes employing ID-based allocation of subsets [15], [14], referred to as random preloaded subsets (RPS) in the rest of this paper, are defined by two parameters (ξ, k) , where $\xi < 1$. The KDC chooses an indexed set of $P = k/\xi$ secrets $\mathbb{S} = \{K_1, K_2, \dots, K_P\}$. A public pseudo-random function $F()$ is employed [15] for ID-based allocation of a subset of $k = \xi P$ secrets to every entity. Thus for an entity with ID A , the set of secrets assigned are

$$\mathbb{S}_A = \{K_{A_1}, \dots, K_{A_k}\} \text{ where } F(A) = \{A_1, \dots, A_k\}, \quad (3)$$

and $1 \leq A_i \leq P$.

Any two entities (say A and B) will share $\bar{m} = \xi k$ secrets on an *average*. The indices of shared secrets can be determined by computing $F(A) \cap F(B) = \{I_1 \dots I_m\}$. Corresponding to each shared index $j = I_i$, A and B can find an *elementary shared secret* $S_i = K_j$. Both A and B can independently compute the SA K_{AB} as a secure one-way function of *all* m shared secrets $S_1 \dots S_m$, as $K_{AB} = h(S_1, \dots, S_m)$. Alternately, K_{AB} could be computed by just XOR-ing the m elementary secrets together.

An attacker who has access to all secrets from n nodes in the set $\mathfrak{D} = \{O_1, \dots, O_n\}$ where $A, B \notin \mathfrak{D}$ has a finite probability of determining the shared secret K_{AB} between A and B . The probability $p(n)$ of such an event is

$$p(n) = (1 - \xi(1 - \xi)^n)^k \quad (4)$$

The optimal choice of ξ that minimizes k for a desired $p(n)$ is $\xi = \frac{1}{n+1}$ [4], [16]. Correspondingly, for large n we have

$$k \approx ne \log(1/p) \quad m = \xi k \approx e \log(1/p) \quad (5)$$

D. The Cost of Resources

Any security solution demands resources which take the form of computation, bandwidth and storage. Different resources have different costs, which may also vary with time and advances in technology. Efficient security solutions should strive to reduce dependence on expensive resources, perhaps by taking advantage of less expensive resources.

Even while advances in technology lowers the cost of *all* resources in accordance with Moore’s Law, there is an obvious *change in balance* between the cost of resources. A few decades ago, asymmetric cryptographic primitives were deemed expensive even for many desktop computers. While this is obviously not the case today, the explosive growth of low complexity network enabled devices will once again result in a scenario where asymmetric primitives are beyond the reach of *most* devices.

In emerging application scenarios mobile computers are expected to come together to form large *co-operative* networks in which every device will be required to perform some tasks for the *overall good* of the network. For example sensors monitoring vital internal organ functions of Bob may send an alarm, which may have to be relayed by Alice’s mobile phone (and possibly many other such devices) to the closest hospital. Furthermore many devices may also be deployed in an *unattended* fashion. For such co-operative networks to be feasible, an essential requirement is the ability to *trust* every device to act in the intended manner.

Computers that have to be trusted, or computers that may have to be deployed without the possibility of close supervision, need to be tamper-responsive. More specifically, they should provide assurances

against ① tampering of the software executed by such computers and ② exposure of secrets used for authentication of the computers by zeroizing secrets under such attempts. While providing such assurances are *not impossible* now [17], [18] they can be *expensive*.

Arguably, the only way to realize *affordable* trustworthy computers is to *reduce the complexity* inside the trusted boundary. As long as the computational complexity to be borne inside a trusted (protected) boundary is low enough *unconstrained* shielding strategies can be used to prevent such computers from intrusions. The need to dissipate heat (in scenarios where the computational complexity inside the trusted boundary is high) can severely cramp strategies for shielding, and thus render them expensive [17]. Furthermore, the need to conserve battery life will also restrict computation and bandwidth overheads that can be tolerated by mobile devices.

However storage continues to reduce in cost at perhaps a rate faster than predicted by Moore’s Law, *especially* for mobile computing scenarios. Thus while there are compelling reasons to reduce computational overheads, we can afford to take advantage of inexpensive storage resources. Also note that increasing the storage overheads, especially storage *outside* the protected boundary, does not hinder strategies for *shielding* trustworthy computers.

1) *Computational Complexity* \mathcal{C} : In the rest of this paper we denote the computational complexity by \mathcal{C} . However in scenarios where proactive measures are taken to protect secrets, there is also a need to distinguish between the complexity of “sensitive” computations (operations performed with secrets) \mathcal{C}_S and public computations (which do not require operations with secrets) \mathcal{C}_U . Sensitive computations will need to be performed *inside* a tamper-sensitive protected boundary. Thus it is very much desirable to reduce the complexity \mathcal{C}_S to facilitate unconstrained shielding. However, public computations could be performed outside the protected boundary.

For instance, a mobile phone / PDA could employ a secure co-processor or smart card or a smart SIM (subscriber identity module) card for performing operations involving secrets. However the more powerful processor of the mobile phone / PDA could be used for performing public computations. Obviously there are compelling reasons to keep \mathcal{C}_S as low as possible. While we would also like to keep \mathcal{C}_U low, this is not as crucial as keeping \mathcal{C}_S low.

2) *Bandwidth Complexity* \mathfrak{B} and *Storage Complexity* \mathfrak{S} : In most application scenarios where an entity is required to store *multiple* secrets, a common practice is to encrypt them with a single “host master secret” which is afforded extensive protection. All encrypted secrets could then be stored in an encrypted key-ring. In scenarios involving *unattended* devices, the host master secret could be stored in a special well protected register. In scenarios involving attended devices it could be a password known only to the end user. The encrypted key ring itself can be stored in unprotected flash storage devices, or even locations possibly accessed over an insecure network. As an example a trustworthy sensor attached to Bob’s chest could communicate over blue-tooth interface with a PDA (with an SD card) where its key ring is stored.

Public values stored in unprotected locations will need to be authenticated (to prevent unauthorized entities from modifying them). Secrets need to be encrypted *and* authenticated. One important consideration in this respect is \mathfrak{B} , the number of values that have to be *fetches from storage* for computing any pairwise secret. Note that even in scenarios where the values do not have to be fetched over a network, fetching a large number of values from slower bulk-storage devices is not desirable.

3) *Descending Order of Cost*: Based on the discussions above, perhaps a reasonable descending order of cost of various resources is as follows:

- ① \mathcal{C}_S Computations with secrets
- ② / ③ \mathfrak{B} Bandwidth overheads
- ② / ③ \mathcal{C}_U Public computations
- ④ \mathfrak{S} Storage

Thus efficient solutions should strive to reduce the complexities \mathcal{C}_S , \mathfrak{B} and \mathcal{C}_U , even if it implies increasing storage complexity \mathfrak{S} .

TABLE I

COMPUTATION, BANDWIDTH AND STORAGE COMPLEXITY MANDATED FOR n -SECURE SKGS AND (n, p) -SECURE KPSs IN TERMS OF KPS PARAMETERS (UNSHADED ROWS) AND THE VALUES n AND p (SHADED ROWS).

KPS	\mathfrak{C}_S	\mathfrak{C}_U	\mathfrak{B}	\mathfrak{S}
SKGS	k	k	k	$k + 1$
	$n + 1$	$n + 1$	$n + 1$	$n + 2$
RPS	ξk	$k \log_2(1/\xi)$	ξk	k
	$\log(1/p)$	$n \log(1/p) \log(n)$	$\log(1/p)$	$n \log(1/p)$
KSSC	m	$m \log_2(M)$	m	mM
	$\log(1/p)$	$\log(1/p) \log(n)$	$\log(1/p)$	$n \log(1/p)$

E. Complexity of KDSs

The high complexity of operations with secrets renders public key schemes (both certificates based and IBE / IBS schemes) unsuitable. For the basic key predistribution scheme the major source of complexity is storage, influenced by the desired network scale N . The computational and bandwidth overheads are however very low. A single readily available shared secret has to be fetched from a key ring where $\mathcal{O}(N)$ such values are stored. Even while storage is inexpensive, such schemes are far from suitable for network scales upward of hundreds of millions.

For scalable KPSs the complexity is contributed by various factors influenced by the value n - the extent of collusion resistance sought. Various factors that affect the complexity of different KPSs are depicted in Table I in terms of the KPS parameters (unshaded rows), and in terms of the values n (and p) (shaded rows) to realize an n -secure (or (n, p) -secure) KPS.

1) Computation Complexity \mathfrak{C} :

a) Computations with Secrets \mathfrak{C}_S : Blom's SKGS scheme requires $k = n + 1$ finite-field multiplications with secrets. RPS requires m operations with secrets to compute K_{AB} .

b) Public Computations \mathfrak{C}_U : For SKGS computation of $\beta_i, 0 \leq i \leq k - 1$ (see Eq (2)) - involving computation of one modular exponent and k modular multiplications, do not require access to secrets. For RPS $F(A) \cap F(B)$ needs to be computed to determine $m = \xi P$ intersecting indices. For RPS computing $F(A)$ involves generation of $k \log_2(1/\xi)$ -bit pseudo-random integers.

2) Storage \mathfrak{S} and Bandwidth \mathfrak{B} : Blom's SKGS scheme calls for every entity to store $k = n + 1$ secrets. For (n, p) -secure RPS entity needs to store $k \approx 2.71n \log(1/p)$.

a) Bandwidth Overheads \mathfrak{B} : For Blom's scheme all $k = n + 1$ secrets stored by an entity are required for computation of every pairwise secret. However for RPS while an entity may store k secrets, only $m \ll k$ of them are required for computing a pairwise secret like K_{AB} .

F. Unsuitability of SKGS and RPS

Blom's KPS requires substantial computational overheads. Both \mathfrak{C}_S and \mathfrak{C}_U require $\mathcal{O}(n)$ finite-field computations. Thus even for n of the order of a few hundreds, the complexity of Blom's KPS becomes comparable to that of asymmetric primitives. Furthermore \mathfrak{B} , viz., the number of values that need to be fetched for computing any pairwise secret, is also $\mathcal{O}(n)$.

RPS requires more storage overheads than deterministic schemes (by a factor $\log(1/p)$). While RPS demands significantly lower complexity of operations with secrets \mathfrak{C}_S and bandwidth overheads \mathfrak{B} the computational overheads \mathfrak{C}_U becomes the bottle-neck for RPS. Note that each entity has to first evaluate the public function $F()$ to determine which of the m (of k) secrets need to be used for computing K_{AB} . The complexity \mathfrak{C}_U of the public function is $k = \mathcal{O}(n \log(1/p))$.

Consider the example of the RPS scheme with parameters $\bar{m} = 121$, $\xi = 2^{-14}$, and $k = 121 \times 2^{14} = 1,982,464$, for which $p(16384 = 2^{14}) < 2^{-64}$. The storage complexity is slightly over 15 MB (if each secret is 64-bits long). Evaluating the public function $F(A)$ amounts to generating k 14-bit pseudo-random values (or $14k$ random bits). More specifically, determining the m shared indices by executing

$F(A) \cap F(B)$ requires (pseudo-random) generation of $2k \times 14 \approx 55.5$ million bits. Thus while the storage overheads (15 MB) is acceptable, the complexity of the public function becomes unacceptably high for large n - even if the computation is performed outside a trusted boundary.

III. KEY SUBSET AND SYMMETRIC CERTIFICATES (KSSC)

The primary motivation for the KSSC, a probabilistic KPS proposed in this section, is to overcome the limitation of the complexity \mathfrak{C}_U of the public function. For KSSC the achievable security is limited only by storage - the least expensive of resources. Furthermore, as we shall see in the rest of this section, compared to RPS KSSC demands

- 1) substantially lower \mathfrak{C}_U ; and
- 2) lower \mathfrak{C}_S , \mathfrak{B} and \mathfrak{S} for achieving (n, p) -security.

A. Symmetric Certificates

The concept of symmetric key certificates (SC) was first suggested by Davis and Swick [5] in the context of Kerberos. From a broad perspective an SC is derived by binding some arbitrary descriptor $D \in \{0, 1\}^*$ (a bit-string of arbitrary length) with a secret key K through a one-way function. For example

$$K^D = h(K, D) \quad (6)$$

is an SC *derived* from K . Note that an SC is derived in the same manner as a hashed message authentication code (HMAC). Furthermore, like HMACs, SCs can be truncated (say only 64 LSBs of the 160-bit hash is retained). However unlike HMACs, SCs are *treated as secrets*. The SC is only privy to the issuer (the entity with secret K) and the entity receiving the SC K^D .

As an example of the utility of SCs, consider a scenario where a server S wishes to bestow upon Alice (A), certain privileges described (for example) by a string $P_A = \text{“Bearer is authorized access the print server from 9 to 5 pm between 8-1-07 and 12-1-07.”}$ The server S can issue a secret $K^A = h(P_A, K_S)$, where K is a secret privy only to the server S . The SC K^A is produced by Alice along with the string P_A whenever she wishes to use her privilege. Note that a server S may assign such privileges to numerous entities like Alice. With the use of symmetric certificates the server S does not have to keep track of *what* privileges were assigned to *whom*. For this reason, Davis and Swick [5] saw symmetric certificates as a “convenient way of issuing memorandums to oneself¹.”

B. KSSC

KSSC is defined by two parameters (m, M) . The KDC chooses

- 1) a set of $k = mM$ secrets $\mathbb{S} = \{K(i, j)\}$, $1 \leq i \leq m, 1 \leq j \leq M$,
- 2) a hash function $h()$, and
- 3) a set of m public functions $f_i()$, $1 \leq i \leq m$

The public functions $f_i()$ take IDs of entities (say 128-bit quantities) as inputs and outputs a uniformly distributed random number between 1 and M (the output is a pseudo-random $\log_2 M$ -bit number).

Every entity is assigned a subset m of the $k = mM$ keys chosen by the KDC, and a set of $k = mM$ symmetric certificates (SC). For an entity with ID A the KDC computes $a_i = f_i(A)$, $1 \leq i \leq m$. The entity A is then assigned a set of m secrets \mathbb{S}_A and a set of mM SCs \mathbb{C}_A bound to the ID A where

$$\begin{aligned} \mathbb{S}_A &= \{K(1, a_1), K(2, a_2), \dots, K(m, a_m)\} \\ \mathbb{C}_A &= \{K_A(i, j) = h(K(i, j), A)\}, 1 \leq i \leq m, 1 \leq j \leq M. \end{aligned}$$

The m secrets \mathbb{S}_A is a subset of the mM secrets chosen by the KDC. The mM SCs \mathbb{C}_A are derived from the mM KDC secrets using the one-way hash function $h()$. More generally, $K_A(i, j)$ can be a truncated hash of $h(K(i, j), A)$ where only (say) $t = 64$ LSBs of a 160-bit hash $h()$ is retained.

The SCs assigned to an entity A are stored *encrypted* in bulk storage. The m secrets \mathbb{S}_A (we shall see that m is typically a few tens) could however be stored in a cache memory.

¹In the context of Kerberos, strings like P_A describe privileges, and the combination of P_A and the SC K^A constitute a *Kerberos ticket*.

1) *Computing Pairwise Secrets*: Two entities A and B can derive up to $2m$ common elementary shared secrets

$$S_i^A = K_A(i, b_i), 1 \leq i \leq m \text{ and} \quad (7)$$

$$S_i^B = K_B(i, a_i), 1 \leq i \leq m \quad (8)$$

Note that the elementary shares are SCs

$$\begin{aligned} \{S_i^A, 1 \leq i \leq m\} &= \mathbb{C}_B^A \subset \mathbb{C}_B \\ \{S_i^B, 1 \leq i \leq m\} &= \mathbb{C}_A^B \subset \mathbb{C}_A. \end{aligned} \quad (9)$$

Entity A can find m elementary shares \mathbb{C}_B^A among its mM SCs \mathbb{C}_A . The indices of the particular m SCs to be used can be determined by computing $b_i = f_i(B), 1 \leq i \leq m$. Entity A can *compute* the m elementary shares \mathbb{C}_B^A using its m secrets. Similarly B has \mathbb{C}_A^B as a subset of its SCs and can compute \mathbb{C}_A^B . All $2m$ elementary shares are used to derive the SA (pairwise secret) K_{AB} as

$$K_{AB} = h(\mathbb{C}_B^A, \mathbb{C}_A^B), \quad (10)$$

or simply XOR-ing all $2m$ elementary shares.

C. Security Analysis

Note that the SCs assigned to any entity do not reveal any information about the secrets or the SCs of *other* entities. While a group of n colluding attackers (that does not include A and B) cannot gain access to the SCs of A and B , they can pool their secrets together to determine $\mathbb{S}_A = \{K(1, a_1) \cdots K(m, a_m)\}$ and $\mathbb{S}_B = \{K(1, b_1) \cdots K(m, b_m)\}$, from which the required SCs (the $2m$ elementary shares) can be computed.

The probability that an entity C has been assigned the secret $K(i, a_i)$ (or the probability that $c_i = f_i(C) = a_i = f_i(A)$) is $1/M$. The probability that a particular elementary share is *not* included in the attacker's pool (collected from n entities) is

$$\epsilon_A = (1 - 1/M)^n \approx e^{-n/M}, \quad (11)$$

and the probability that the attackers pool of secrets includes *all* the required $2m$ secrets is

$$p(n) = (1 - \epsilon_A)^{2m} \approx (1 - e^{-n/M})^{2m}. \quad (12)$$

1) *Complexity*: The computational complexity for the scheme is $\mathcal{O}(m)$. Evaluation of K_{AB} involves

- 1) \mathfrak{C}_U - computation of m public functions $f_i(\cdot)$ and
- 2) \mathfrak{C}_S - m hash function evaluations to compute m S_i s.
- 3) \mathfrak{B} - m SCs to be fetched from bulk storage.

Most often the public function evaluations like $a_i = f_i(A)$ are computed for all $1 \leq i \leq m$. The hash function $h(\cdot)$ can be used for realizing the public functions efficiently. If, for example, $m = 32$ and $\log_2 M = 15$, we need to generate $m \log_2 M = 15 \times 32 = 480$ pseudo-random bits. If we use a 160-bit hash function $h(\cdot)$ the 160×3 bits of $[h(A, 1) \parallel h(A, 2) \parallel h(A, 3)]$ can be interpreted as $[a_1 \parallel \cdots \parallel a_m]$. Thus for KSSC

$$\begin{aligned} \mathfrak{C}_S &= m & \mathfrak{C}_U &= m \log_2 M \\ \mathfrak{B} &= m & \mathfrak{S} &= m + Mm \end{aligned} \quad (13)$$

D. Choice of Parameters

A quick inspection of Eq (12) reveals that for a choice of $n \propto M$, we have $m \propto \log(1/p)$. It is very important to note that the computational complexity (both \mathfrak{C}_S and \mathfrak{C}_U) and bandwidth overheads \mathfrak{B} are *independent* of n . We can thus increase n to any extent by increasing M , which will result in more storage required for SCs. Thus the extent of collusion resistance n that can be realized is constrained *only* by available storage.

1) *Minimizing Storage*: We can rewrite Eq (12) as

$$k = mM = \frac{n \log(1/p)}{-2\theta \log(1 - e^{-\theta})}, \text{ where } \theta = \frac{n}{M}. \quad (14)$$

If we intend to minimize the storage $k \approx mM$ (required for the SCs) for realizing an (n, p) -secure KSSC, we need to maximize $-\theta \log(1 - e^{-\theta})$, which occurs for $(\theta = \log(2)) = \theta^*$. The optimal choice of parameters that minimizes k is thus

$$\left. \begin{aligned} m^* &= \frac{\log(1/p)}{2 \log 2} \\ M^* &= \frac{n}{\log 2} \end{aligned} \right\} \Rightarrow k^* = \frac{n \log(1/p)}{2 \log^2 2} \quad (15)$$

a) *Numerical Example*: KSSC with parameters $m = 32$ and $M = 2^{15}$ is the storage optimal design to meet the criteria $p(22,500) < 2^{-64}$. Such a scheme requires each entity to store $m = 32$ secrets and $mM = 2^{20}$ (a million) SCs. For 64 bit SCs the storage required is 8 MB. An attacker who has access to secrets assigned to over 22500 entities can determine only a fraction 2^{-64} of pairwise secrets (between entities that are not part of the n colluders). An attacker who has exposed secrets from $n' = 42,200 > n$ entities can expose one in one-billion pairwise secrets (or $p(42200) \approx 2^{-30}$).

While for Blom's scheme and RPS some complexity factors increase linearly with n (see Table I), for KSSC *only* storage² is linear in n . As an example, while $p(22500) \approx 2^{-64}$ for KSSC with $m = 32, M = 2^{15}$ (storage 8 MB), doubling M to 2^{16} will result in a scheme for which $p(45000) \approx 2^{-64}$. The storage complexity mM is doubled to 16 MB. The complexity of the public function increases marginally. While the former scheme requires pseudo-random generation of $m = 32$ 15-bit integers, the latter scheme ($m = 32, M = 2^{16}$) requires pseudo random generation of $m = 32$ 16-bit integers. If we desire a KSSC scheme with $p(100000) \approx 2^{-64}$, a little less than 36 MB of storage is required for every entity.

2) *Computation-Storage Trade-offs*: Even while KSSC requires very low computational overheads (about m hash function evaluations) it may still be advantageous to reduce m further (by increasing M and $k = mM$) as m has bearings on the complexities $\mathfrak{C}_S, \mathfrak{C}_U$ and \mathfrak{B} , while M affects only storage (the least expensive resource).

If we desire to reduce m by a factor a (or $m' = m/a$) we need to chose $M' > M$ such that

$$\begin{aligned} p(n) &= (1 - e^{-\frac{n}{M}})^{2m} = (1 - e^{-\frac{n}{M'}})^{2m'} \Rightarrow \\ e^{-\frac{n}{M'}} &= 1 - 1/2^a \end{aligned} \quad (16)$$

Thus

$$\left. \begin{aligned} m' &= \frac{\log(1/p)}{2a \log 2} \\ M' &= \frac{n}{\log(1 - 1/2^a)} \end{aligned} \right\} \Rightarrow k' = k^* \frac{\log(1 - 1/2)}{a \log(1 - 1/2^a)}. \quad (17)$$

The table below depicts the tradeoffs involved in reducing m (by a factor a) vs the corresponding increase in total storage s :

a	2	3	4	5	8
k'/k^*	1.204	1.730	2.685	4.366	22.137

Thus decreasing m by a factor of 3 ($a = 3$) calls for increasing storage k by a factor 1.73 (and increasing M by a factor 3×1.73). Decreasing m from 32 to 12 (in the numerical example illustrated in Section III-D.1.a) calls for a four-fold increase in M and a 1.5 fold increase in s . In other words, both PLM schemes viz., $(m = 32, M = 2^{15})$ and $(m = 12, M = 2^{17})$, meet the requirement $p(22500) < 2^{-64}$.

²The complexity of the public function has a $\log_2 n$ dependency on n .

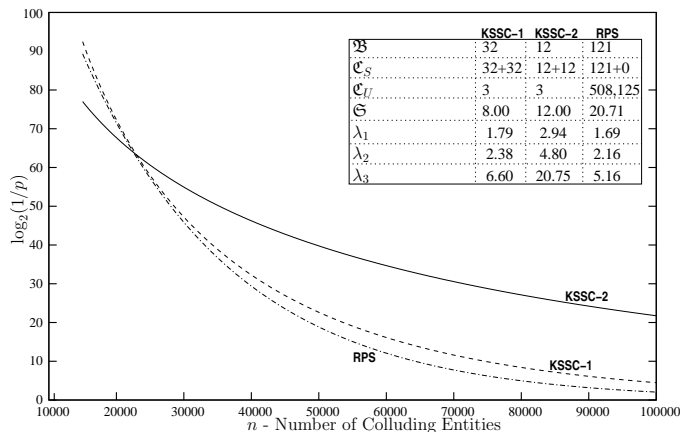


Fig. 1. Performance of RPS and KSSC designed to meet the criterion $p(22500) \leq 2^{-64}$. Note that all plots intersect at the point $n = 22500$ and $\log_2(1/p) = 64$.

E. Performance Comparison

A performance comparison of RPS and KSSC designed to meet the criterion $p(n) < 2^{-64}$ for $n \leq 22500$ (or $-\log_2(p(n)) > 64$ for $n \leq 22500$) is depicted using plots of $-\log_2 p$ vs n in Figure 1. The enclosed table in Figure 1 also provides *quantitative* measures of various complexities associated with the KPSs and the rate of degradation of security (even while the latter is evident from the plot).

In the table in Figure 1 the computational complexity is broken down into complexity of operations with secrets \mathfrak{C}_S and complexity of public operations \mathfrak{C}_U . The value \mathfrak{C}_S is itself broken down into two components. For example, for RPS the $m = 121$ secrets fetched from storage (where they are stored encrypted) have to be decrypted to yield the m elementary shares S_i . For KSSC-2 $m = 12$ SCs have to be fetched from storage and decrypted. In addition $m = 12$ SCs have to be computed using the 12 secrets stored in cache memory.

The complexity \mathfrak{C}_U is indicated in terms of the number of 160-bit hash function evaluations. For RPS, and KSSC computation of public functions ($F()$ and $f()$ respectively) calls for pseudo-random generation of bits. A 160-bit hash function could be used to generate 160 pseudo-random bits. The storage complexity \mathfrak{S} is shown in MBs, assuming 64-bit values (for all schemes).

Apart from demanding very low *expensive* overheads compared to RPS, KSSC also has the least rate of degradation of security, which can further be improved by trading off some storage efficiency (KSSC-2, higher mM) and simultaneously reducing the computational and bandwidth overheads.

IV. PRACTICAL DEPLOYMENT ISSUES

Large scale networks do not typically *start* as large scale networks. A potential large scale network may start with a few thousands or even a few hundred thousands and may grow to several billions. With the large storage capabilities of current mobile devices, even for network sizes of *millions* non-scalable KDSs which require very low computational overheads are well suited. After all, storing 5 million 128-bit secrets requires only 64 MB of storage.

In this section we argue why an elegant KDS proposed by Leighton and Micali in [3] is a substantially more appealing alternative to the basic KPS for small scale networks. We then argue that a combination of LM-KDS and KSSC has many compelling advantages for practical deployments.

A. Network Model

We shall assume that entities are *inducted* into the network by the KDC (perhaps controlled by the network operators) by providing them with secrets corresponding to some ID. We shall represent the

entities as A (Alice), B (Bob), C (Charlie) etc. Values like A, B, C could be 128-bit hashes of long descriptive real-life identities. Let us further assume that

- 1) the *only* time that the entities have access to the KDC is during their induction into the network, and
- 2) that entities can be inducted at *any* time.

Any two inducted entities should thereafter be able to ① establish a shared secret (say 128-bit strong), and ② determine each other's real life identities.

Obviously, the most important criteria to be considered is the network scale. We shall use the following three distinct measures for scale of the KDS:

- 1) N - the size of the ID-space, as each entity requires a unique ID;
- 2) $N_a(t)$ - the total number of entities inducted *before* time t ; and
- 3) N_m - the maximum number of entities that are expected to be inducted;

The value N_m is the “network operators conservative guess” of $N_a(t)$ for $t \rightarrow \infty$.

If the basic KPS is used it may appear at first sight that the KDC can choose a master secret K and $K_{AB} = K_{BA}$ used for mutual authentication of A (Alice) and B (Bob) can be $K_{AB} = K_{BA} = h(K, A, B) \oplus h(K, B, A)$. However such an approach is *not* possible when entities are inducted into the network *asynchronously*. For example, if A joins the network before B , the network operator does not even know the ID B that will be assigned to an entity Bob in the future. Thus there is no way for the KDC to provide Alice with the value K_{AB} .

One option is to associate each entity with two IDs. For example, Alice, say the i^{th} entity to be inducted into the network, can be associated with a real-life identity like A (128-bit ID) and a “KDS identity” i . Bob (who is inducted after Alice) is associated with IDs B and $j_1 > i$. As we need only N_m different KDS IDs, the IDs like i and j_1 could be $\log_2 N_m$ bits long.

Alice can be provided with ① N_m secrets K_{ij} and ② N_m “commitments” X_j , $1 \leq j \leq N_m$ (or a single commitment X derived using a Merkle hash tree [19] with X_j s as leaves). Each commitment can serve as a public value for a one-time-signature [20] scheme. Thus a commitment X_i can be used to bind the values A and i using a certificate $\langle A, i \rangle_{X_i}$. While A and B share a secret K_{ij_1} , they will need to exchange certificates to determine their real-life identities.

B. The Leighton-Micali KDS

In [3] Leighton and Micali proposed an elegant KDS, the LM-KDS, as an alternative to Kerberos. In LM-KDS a KDC chooses a master key K and a hash function $h(\cdot)$. Entity A is provided with the secret $K_A = h(K, A)$. To facilitate mutual authentication of A and B , a public value

$$P_{AB} = h(K_A, B) \oplus h(K_B, A) = P_{BA} \quad (18)$$

is used. When A desires to establish a session secret with B , A approaches the *on-line* KDC to get the public value P_{AB} . The shared secret between A and B is computed by A as

$$K_{BA} = h(K_A, B) \oplus P_{AB} = h(K_B, A). \quad (19)$$

Note that B can also compute the shared secret *without* using the public value P_{AB} .

For small-scale networks (small N) the LM-KDS can be used to provide *ad hoc* establishment of SAs if every entity stores one secret and $N - 1$ public values. More specifically, while in the basic KPS each entity requires to store $N - 1$ secrets, for the LM-KDS (used *without* an online KDC), each entity stores one secrets and a maximum of $N - 1$ public values. We shall now see why LM-KDS is a substantially more appealing alternative for a small scale network model described in Section IV-A.

1) *Using LM-KDS*: When the LM-KDS is used, the KDC chooses a master secret K . Let Alice be the i^{th} entity to be inducted into the network. Alice (A) is assigned the secret $K_A = h(M, A)$. In addition, Alice receives $i - 1$ values $\langle I_j, P_{AI_j} \rangle$, $1 \leq j \leq i - 1$, where

$$P_{AI_j} = h(K_{I_j}, A) \oplus h(K_A, I_j) = P_{I_jA}, \quad (20)$$

and I_j 's are identities of entities who were inducted into the network *before* Alice. Thus Bob (B) who joins the network after A will store *all* public values for establishing a secret with *all* entities who joined before B (including obviously, Alice). Alice can compute $K_{AB} = h(K_A, B)$ and Bob can compute $K_{AB} = h(K_B, A) \oplus P_{AB}$. Thus mutual authentication of any two entities based on their real-life identities is possible irrespective of *when* the entities joined the network, *without* requiring overheads for exchanging certificates.

2) *Basic KPS vs LM-KDS*: In practical scenarios the number of entities who have *actually* joined the network till a time t , viz., $N_a(t)$ may be *substantially* lower than the ‘‘conservative guess’’ of the maximum possible number of entities. If the basic KPS is used each entity needs $\mathcal{O}(N_m)$ storage *and* use one-time signatures in addition.

With LM-KDS an entity inducted into the network at time t requires to store $N_A(t)$ values like $\langle I_j, P_{AI_j} \rangle$ (or $N_a(t)$ 32-byte values if IDs and public values are 128-bits long). An entity who joins the network after ten thousand other entities requires 312 KB of storage. The millionth node to join the network will require about 32 MB of storage.

That the storage required for any entity grows linearly depending on the time at which it joins the network is intuitively appealing. After all with reducing costs of storage with time, newer entities (say entities inducted in the year 2009) can afford to store more values than older entities (say inducted in 2007). Furthermore, that the stored values are *public* values that do not need as much protection as secrets is also a desirable feature. Public values could even be stored in read-only media to prevent accidental or intentional modifications.

LM-KDS also lends itself readily to employing multiple independent escrows. For instance in a scenario where two KDCs with master secrets ${}_1K$ and ${}_2K$ are used, A can receive secrets ${}_1K_A = h({}_1K, A)$ and ${}_2K_A = h({}_2K, A)$. Thus establishment of SAs require two public values of the form ${}_iP_{AB} = h({}_iK_A, B) \oplus h({}_iK_B, A)$. Each node however needs to store only one value $P_{AB} = {}_1P_{AB} \oplus {}_2P_{AB}$. If only B has access to the public value P_{AB} the pairwise secret K_{AB} can be computed as

$$K_{AB} = \begin{cases} h({}_1K_A, B) \oplus h({}_2K_A, B) & \text{by } A \\ h({}_1K_B, A) \oplus h({}_2K_B, A) \oplus P_{AB} & \text{by } B \end{cases} \quad (21)$$

Apart from eliminating the need for certificates *and* requiring substantially lower storage overhead, perhaps the most compelling advantage of LM-KDS is that there is *no need to place a hard limit* on the value N_m . In other words the network operators do not have to ‘‘guess’’ the potential network scale. If at some time t' the number of entities $N_a(t')$ goes beyond practical storage capabilities, one can continue to use the LM-KDS in the ‘‘Kerberos mode’’ (the way it was intended in [3]) where the KDC is online to provide public values on demand.

C. Growing Pains

While LM-KDS can facilitate ad hoc authentication for small scale networks, ad hoc SAs will no longer be feasible as the network size grows to very large extents.

1) *Ad Hoc vs Planned SAs*: It is important to note that ad hoc SAs between two entities A and B are required *only* in scenarios where

- 1) neither A nor B have *a priori* knowledge of their need to communicate with each other, *and*
- 2) it is impractical for either node to contact the KDC (say over the Internet).

An example of such a scenario is the case when A and B happen (purely by accident) to be neighbors in a multi-hop ad hoc network where it is impractical for either node to communicate with a KDC. Another

example is a scenario where A and B meet accidentally in a remote location where no connection to the Internet exists. For *planned* interactions, or scenarios where it is practical for at least one node to contact the KDC, there is no *justification*, nor the *need*, for ad hoc SAs based on collusion susceptible KPSs. The LM-KDS (used in Kerberos-mode) is more than adequate for this purpose.

The use of scalable KPSs like KSSC is mandated only for unplanned interactions where access to the KDC is impractical. In the rest of this section we shall argue why probabilistic KPSs, and more specifically KSSC, is well suited to facilitate this requirement.

D. Probabilistic KPSs for Ad Hoc SAs

Apart from graceful degradation of security with increased number of compromised nodes, probabilistic KPSs (like RPS and KSSC) have other substantial advantages over deterministic KPSs like Blom's SKGS.

1) *Low Complexity Hardware*: Implementation of RPS and KSSC requires only block-cipher and / or hash operations. A very low complexity cryptographic co-processor / smart card / SIM card, equipped with a single reusable hardware block cipher *or* hash function³ can be used for *all* computations involving secrets.

Implementation of finite field arithmetic required for Blom's SKGS can be more expensive. Furthermore, a block cipher / hash function is required *in any case* to perform computations (encryption of messages and computing HMACs) *using* pairwise secrets facilitated by the KPS.

2) *KDC Complexity*: For all KPSs the number of secrets P to be chosen by the KDC is of very little consequence. As the KDC has unrestricted freedom in choosing the core secrets a single master secret can be used to derive all P values using a secure one-way function. However what is an issue is the complexity required for computing all secrets to be provided to an entity with some ID (say A).

For SKGS the KDC has to compute $\mathbb{S}_A = \mathbf{d}^A = \mathbf{D}\mathbf{g}_A$ which requires $(n+1)^2$ finite-field multiplication operations, which can become prohibitively expensive for large n . For probabilistic KPSs the complexity involved for computing \mathbb{S}_A is $\mathcal{O}(n \log(1/p))$ hash function evaluations.

3) *Renewal and Escrow*: The (n, p) -secure probabilistic KPSs lend themselves readily to ① *seamless* renewal, and ② simple strategies for employing multiple *independent* escrows (KDCs).

For Blom's KPSs changing even one of the P secrets (any value of the symmetric matrix \mathbf{D}) chosen by the KDC will result in modification of *every* secret assigned to *every* entity. So renewal involves complete modification of all secrets assigned to every entity. This may be very difficult to achieve without *interrupting* the operation of the deployment. Furthermore, if two SKGS schemes are deployed in parallel (say controlled by two independent KDCs, and each entity receives $\mathcal{O}(n)$ secrets from each KDC) the resulting KPS, where SAs from both KPSs are used to establish pairwise secrets, is still only n -secure. Thus simple strategies for increasing the number of escrows will cause increase in complexity proportional to the number of escrows.

However for (n, p) -secure KPSs the situation is very different. Two (n, p) -secure KPSs can be combined to yield an (n, p^2) -secure KPS. For example, (n, p) -secure RPS with parameters (ξ, k) can actually be t parallel deployments of (n, p_i) -secure RPS schemes with parameters (ξ, k_i) where $1 \leq i \leq t$, $\prod_{i=1}^t p_i = p$, and $\sum_{i=1}^t k_i = k$. Thus parallel deployments (controlled by independent KDCs) can be realized *without* any loss of efficiency. To facilitate seamless renewal $t-1$ of the t systems could be used during the finite period required for renewing the secrets of one of the t systems. Similarly an (n, p) -secure KSSC with parameters (m, M) can actually be m parallel deployments controlled by m KDCs.

4) *Multi-path Diversity*: Another advantage of (n, p) -secure KPSs is that in applications like MANETs and multi-hop sensor networks, nodes can make use of *multi-path* diversity [13] to improve the collusion resistance. For example in a scenario where nodes A and B have three paths, say $A \rightarrow B$, $A \rightarrow C \rightarrow B$ and $A \rightarrow D \rightarrow B$, A can send three independent components of a session secret over the three independent paths. An (n, p) -secure RPS is rendered $(n, 4p^3)$ -secure under this scenario. Note that in order to break the session secret the attacker has to compromise K_{AB} and $(K_{AC}$ or $K_{CB})$ and $(K_{AD}$ or $K_{DB})$. As the

³As a block-cipher can be used as a hash function and a hash function can be used as a block cipher [21]

probability that any SA can be compromised is p (by an attacker who has exposed secrets from n nodes) the probability that the attacker can compromise the session secret between A and B is $p \times 2p \times 2p = 4p^3$. Deterministic n -secure schemes cannot take advantage of multi-path diversity due to the catastrophic onset of failure - either all links (SAs) are safe or all links are compromised.

5) *Attackers with Specific Intentions*: Almost all the advantages of probabilistic KPSs (over deterministic KPSs) stem from the fact that only a small fraction of the stored secrets ($m \ll k$) need to be *used* for evaluating any SA. This fact also has a subtle disadvantage, viz., lower resistance to attacker with *specific intentions*. We shall now see why this is *not* a disadvantage *as long as KPS SAs are used only for unplanned SAs*.

Consider a scenario where an attacker desires to gain access to a specific secret K_{XY} used for mutual authentication of X and Y . With access to K_{XY} the attacker can either impersonate X for purposes of fooling Y or vice-versa. If the attacker's strategy is to just compromise as many nodes that she can gain access to, and if she has no way of *intelligently* choosing which nodes to compromise, a very large number of nodes have to be compromised to expose K_{XY} . For example, for KSSC with parameters $m = 12$ and $M = 2^{17}$, an attacker who has exposed secrets from close to half a million entities has an even chance ($p = 0.5$) of determining the required K_{XY} . However an attacker with very *specific* intentions does *not* have to actually expose secrets from *every* node. All she has to do is *identify* half a million nodes she *could* gain access to. Out of these nodes, she has to expose the required $2m$ secrets (for computing K_{XY}) from at most 24 nodes.

However, as long as KPS SAs are employed *only* for ad hoc (unplanned) SAs, attackers have very little to gain by compromising very specific SAs. After all if X and Y do not foresee their need to interact at some point in time (and hence use KPS SAs when they actually do), it is unlikely that an attacker may be "lurking in wait" for such an eventuality. In addition, whenever it is convenient to do so, multi-path diversity can be used to further reduce the risk posed by such an attacker. Note that in such cases the attacker will require knowledge of *multiple* pairwise secrets.

V. RELATED WORK

Canetti et al [4] proposed an elegant broadcast authentication (BA) scheme which employs a concept very similar to symmetric certificates⁴ derived from the secrets chosen by the KDC. The scheme in [4] supports any number of "external entities" to broadcast authenticated messages.

In [4] the KDC chooses k secrets $K_1 \cdots K_k$. "Internal entities" (say $A, B, C \cdots$, capable of verifying broadcasts) are provided a subset of $m \ll k$ secrets on an average - every entity is assigned any of the k secrets with a probability $\xi = m/k$. The KDC issues k SCs to every "external entity" (who are not provided with *any* of the k KDC secrets) which are derived from the k secrets of the KDC through a one way function. For example an external entity Λ is provided with SCs $K_i^\Lambda = h(K_i, \Lambda)$, $1 \leq i \leq k$. Along with a message \mathcal{M} , broadcast source Λ appends k HMACs $H_i = h(\mathcal{M}, K_i^\Lambda)$. Any entity can verify m of the k appended HMACs on an average (every entity can compute m of the k SCs provided to Λ using their subset of m secrets). The probability $p(n)$ that an attacker who has access to the secrets stored in n nodes can impersonate any external source for fooling a particular entity A (or the probability that the pool of secrets accumulated from n nodes includes all m secrets of A) is

$$p(n) = (1 - \xi(1 - \xi)^n)^k \quad (22)$$

One of the primary motivations of KSSC stems from the realization that the BA scheme in [4] is in fact *more suitable for establishing shared secrets* between an external entity Λ and any internal entity (with m secrets). Note that when used for BA we *cannot* afford to increase k to very large extents as k is the number of HMACs appended. However when used for facilitating shared secrets k is only the number of SCs that need to be *stored*. Only m of the k SCs need to be employed for computing a shared secret $K_{\Lambda A}$.

⁴The authors however refer to the set of k secrets chosen by the KDC as "primary keys" and the SCs as "secondary keys."

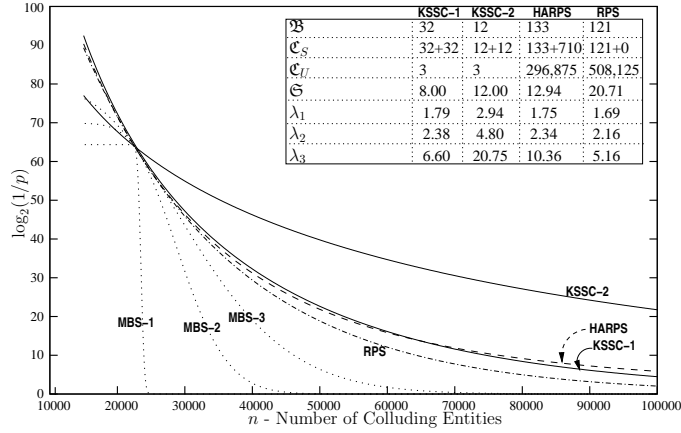


Fig. 2. Performance of several KPSs designed to meet the criterion $p(22500) \leq 2^{-64}$.

In KSSC “internal entities” are assigned exactly m secrets and k SCs. Any two entities share (or can compute) a set of $2m$ common SCs.

A. Other Probabilistic KPSs

Several other probabilistic KPSs for pairwise secrets have also been proposed in the literature.

1) *HARPS*: In hashed random preloaded subsets (HARPS) [16] with parameters (ξ, k, L) , the KDC chooses an indexed set of $P = k/\xi$ secrets $\mathfrak{S} = \{K_1 \cdots K_P\}$, and a public functions $F_H(\cdot)$. The public function $F_H(A) = \{(A_1, a_1), (A_2, a_2), \dots, (A_k, a_k)\}$ determines the indices of k (of P) secrets assigned to entity A and the “hash depth” of each secret. More specifically, the secret K_{A_i} is repeated hashed a_i times before it is assigned to A (say $a_i K_{A_i}$).

Two entities A and B compute their $m = \xi k$ shared indices as in RPS, and the respective hash depths for each shared index. For some shared index (say j) B has the secret $b_j K_j$ and A has the secret $a_j K_j$. The elementary shared secret S_i corresponding to the shared index is $\max(a_j, b_j) K_j$. If $a_j > b_j$ then entity A already has the elementary shared secret. Entity B needs to repeatedly hash its secret $b_j K_j$, $a_j - b_j$ times. As the hash depths are uniformly distributed between 1 and L , for any shared index the average difference in hash depth is $L/3$. However, as only one of the two entities need to perform the hashing, a total of $mL/6$ hashes have to be computed by each entity.

While HARPS requires slightly more computational overheads \mathfrak{C}_S (due to the need for $mL/6$ hashes), it retains all the advantages of RPS discussed in Section IV-D. Furthermore, it requires lower k compared to RPS (and hence lower complexity of the public function $F(\cdot)$) and sports more graceful degradation of security than RPS (see Figure 2). More specifically, even while the complexity of the public function involves generation of $k(\log_2(1/\xi) + \log_2 L) = k \log_2 L/\xi$ bits (instead of $k \log_2(1/\xi)$ for RPS, the lower value of k results in lower complexity of the public function $F(\cdot)$). For HARPS with $L = 32$ designed to meet $p(22500) < 2^{-64}$, the minimal value of $k \approx 1.7e^6$ for $\xi = 0.000077$ (instead of $k \approx 2.7e^6$ for RPS). The public function computation will require 296,875 160-bit hash computations (instead of 508,125 for RPS).

2) *MBS*: Multi-space Blom’s schemes involving combinations of RPS with Blom’s schemes have been independently proposed in [22] and [23]. The scheme proposed in [22] combines RPS with Blom’s SKGS scheme. The scheme proposed by [23] combines RPS with Blom’s polynomial scheme [7].

MBS is characterized by three parameters (ξ, k, n_b) . The KDC chooses $P = k/\xi$ independent instances of n_b -secure SKGS schemes. Each entity receives $k_b = n + 1$ secrets corresponding to $k = \xi P$ of those schemes. Thus the total of kk_b values are provided to each entity (storage complexity $\mathfrak{S} = k(n_b + 1)$). Any two entities will share $\bar{m} = \xi k$ instances of Blom’s KPSs on an average. In each shared scheme the two entities can discover an elementary shared secret S_i .

The main motivation for MBS seems to be the extension of SKGS to realize more graceful degradation of security. Simultaneously, MBS gains many of the other advantages of probabilistic KPSs discussed in Section IV-D (like the ability to take advantage of multi-path diversity and cater for seamless renewal / multiple escrows). However, like Blom’s scheme it mandates use of more complex hardware for finite-field arithmetic, *and* demands substantially higher complexities \mathfrak{C}_S and \mathfrak{B} .

For MBS one has the freedom to choose the parameter ξ depending on the desired rate of degradation of security. An (n, p) -secure MBS scheme with parameters (ξ, k, n_b) can be realized by combining a (n_r, p) -secure RPS with parameters (ξ, k) with an n_b -secure Blom’s scheme where $n_r n_b \approx n$. If we choose $n_r \ll n_b$ the resulting scheme is more “Blom KPS-like” (rapid degradation of security, but more storage efficient). For $n_b \ll n_r$ the resulting scheme is more “RPS-like” (more graceful degradation of security, lower storage efficiency).

For MBS computation of each S_i calls for ① $n_b + 1$ values to be fetched from storage, ② $n_b + 1$ finite-field multiplications with secrets and ③ $n_b + 1$ exponentiation operations using the public values like α and IDs. Thus for MBS $\mathfrak{C}_C = \mathfrak{C}_U = \mathfrak{B} = \xi k(n_b + 1)$. In addition MBS also requires computation of a public function $F(A) \cap F(B)$ (similar to RPS and HARPS) for determining the ξk intersecting indices.

For MBS, once a suitable ξ is chosen, the optimal choice of parameters involves finding a suitable n_b that minimizes $k(n_b + 1)$. For MBS designed to meet $p(n = 22500 = 2^{-64})$ the optimal choice of parameters for $\xi = 2^{-3}, 2^{-10}, 2^{-12}$ are $(n_b = 2940, k = 334)$, $(n_b = 28, k = 49563)$, and $(n_b = 7, k = 224205)$ respectively. The $p(n)$ vs n plots for the MBS schemes are also depicted in Figure 2 (labelled MBS-1 ($\xi = 1/8$), MBS-2 ($\xi = 2^{-10}$) and MBS-3 ($\xi = 2^{-12}$)).

MBS-3 with high ξ ($\xi = 1/8$) has very little advantage over the SKGS scheme in terms of the rate of degradation of security while mandating $\mathfrak{C}_S = \mathfrak{C}_U = \mathfrak{B} = \xi k(n_b + 1) \approx 122787$ which is far greater than the complexity required for SKGS (which would require $\mathfrak{C}_S = \mathfrak{C}_U = \mathfrak{B} = n + 1 = 22501$). For $\xi = 2^{-10}$ (MBS-2) and $\xi = 2^{-12}$ (MBS-3) the complexity reduces to 1404 and 438 respectively. However for low ξ the complexity of the public function $F(A) \cap F(B)$ increases. MBS-2 (MBS-3) requires an equivalent of 6196 (33631) 160-bit hash function evaluations to compute $F(A) \cap F(B)$.

Also note that the sole advantage of MBS compared to RPS (lower storage overheads) is reduced for small ξ . Assuming 64-bit values MBS-2 and MBS-3 require storage of 11.34 and 15.39 MBs respectively (compared to 20.71 MB for RPS and 12.94 MB for HARPS). Even this advantage is reduced in scenarios where large ID-space is required (say 128-bit IDs) as for MBS (and Blom’s scheme) each secret will need to be 128-bits long (as IDs and secrets belong to $\mathbb{Z}_q, q \geq N$). However this is not necessary for KPSs that do not rely finite-field operations (RPS, HARPS, KSSC). While IDs like A and B could be 128-bits long, each secret could be 64 bits long. After all m such secrets are combined to derive the final pairwise secret K_{AB} .

VI. CONCLUSIONS

We proposed an efficient key distribution scheme for facilitating cryptographic authentication of resource constrained devices which are expected to take part in emerging networks. In such evolving application scenarios involving large scale deployments of mobile computing devices, computation and bandwidth overheads render scalable key distribution schemes that rely on asymmetric cryptographic primitives unsuitable. However storage is an inexpensive resource for such devices.

The thesis central to this paper is that the *low cost of storage can be used to offset the main disadvantage of symmetric cryptographic primitives*, viz., constraints on their scalability. We argued that a combination of the KDS proposed by Leighton and Micali (LM-KDS) in [3] used in conjunction with the novel KPS proposed in this paper (KSSC), is an appealing option for emerging large-scale networks. KSSC can take good advantage of inexpensive storage resources to realize such high levels of collusion resistance that its “fragility” may be irrelevant in practice.

We identified different sources of complexity in KPSs which prevent existing KPSs from achieving high levels of collusion resistance. We argued that efficient KPSs should strive to take advantage of less

expensive resources (like storage) and reduce dependence on more expensive resources (computation and bandwidth overheads) to meet this requirement.

We argued that the simple and elegant LM-KDS can be used to cater for ad hoc SAs even when several million entities may need to take part in a network. However when the number of entities goes beyond reasonable storage requirements, LM-KDS cannot be used for facilitating ad hoc SAs as the KDC will be required to be online. In such scenarios KSSC can be used for facilitating ad hoc SAs.

In practice, emerging networks can begin using *only* LM-KDS till some time t_1 where the number $N_a(t_1)$ (the number of entities who have actually been inducted into the network) reaches a few millions. Till this point there is neither the justification, nor the need, for KSSC. However, as the number of inducted entities become large enough to render storage of $N_a(t > t_1)$ values impractical, the network can seamlessly switch to using LM-KDS in the Kerberos-mode for planned interactions, and rely on KSSC SAs for unplanned interactions.

REFERENCES

- [1] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, 21(12), December 1978.
- [2] T. Matsumoto, H. Imai, "On the Key Predistribution System: A Practical Solution to the Key Distribution Problem," pp 185–193. *CRYPTO 1987*.
- [3] T. Leighton, S. Micali, "Secret-key Agreement without Public-Key Cryptography," *Advances in Cryptology - CRYPTO 1993*, pp 456-479, 1994.
- [4] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, "Multicast Security: A Taxonomy and Some Efficient Constructions," *INFOCOMM'99*, 1999.
- [5] D. Davis, R. Swick, "Network Security via Private-Key Certificates," *Proceedings of the 3rd USENIX Security Symposium*, Baltimore, Sep. 1992.
- [6] D. Boneh, M. Franklin, "Identity-based encryption from the Weil pairing," *Advances in Cryptology – Crypto'2001*, Lecture Notes on Computer Science 2139, Springer-Verlag (2001), pp. 213–229.
- [7] R. Blom, "Non-public Key Distribution," *Crypto-82*, pp 231–236, 1982.
- [8] R. Blom, "An Optimal Class of Symmetric Key Generation Systems," *Advances in Cryptology: Proc. of Eurocrypt 84*, Lecture Notes in Computer Science, **209**, Springer-Verlag, Berlin, pp. 335-338, 1984.
- [9] L. Gong, D.J. Wheeler, "A Matrix Key Distribution Scheme," *Journal of Cryptology*, **2**(2), pp 51-59, 1990.
- [10] C.J. Mitchell, F.C. Piper, "Key Storage in Secure Networks," *Discrete Applied Mathematics*, **21** pp 215–228, 1995.
- [11] M. Dyer, T. Fenner, A. Frieze and A. Thomason, "On Key Storage in Secure Networks," *Journal of Cryptology*, **8**, 189–200, 1995.
- [12] L. Eschenauer, V.D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," *Proceedings of the Ninth ACM Conference on Computer and Communications Security*, Washington DC, pp 41-47, Nov 2002.
- [13] H. Chan, A. Perrig, D. Song, "Random Key Pre-distribution Schemes for Sensor Networks," *IEEE Symposium on Security and Privacy*, Berkeley, California, May 2003.
- [14] R. Di Pietro, L. V. Mancini, A. Mei, "Random Key Assignment for Secure Wireless Sensor Networks," *2003 ACM Workshop on Security of Ad Hoc and Sensor Networks*, October 2003.
- [15] M. Ramkumar, N. Memon, R. Simha, "Pre-Loaded Key Based Multicast and Broadcast Authentication in Mobile Ad-Hoc Networks," *Globecom-2003*.
- [16] M. Ramkumar, N. Memon, "An Efficient Random Key Pre-distribution Scheme for MANET Security," *IEEE Journal on Selected Areas of Communication*, March 2005.
- [17] S.W. Smith, S. Weingart, "Building a High-Performance Programmable Secure Coprocessor," *IBM Technical Report RC21102*, Feb 1998.
- [18] J.D Tygar, B. Yee, "Dyad: A system for Using Physically Secure Coprocessors," *Technological Strategies for the Protection of Intellectual Property in the Networked Multimedia Environment*, pp 121–152, 1994.
- [19] R.C. Merkle "Protocols for Public Key Cryptosystems," In *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, 1980.
- [20] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," In *Advances in Cryptology, Crypto 87*.
- [21] B. Schneier, *Handbook of Applied Cryptography*, Second Edition, John Wiley and Sons Inc., 1996.
- [22] W. Du, J. Deng, Y.S. Han, P.K. Varshney, "A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks," *Proceedings of the 10th ACM Conference on Computer and Communication Security*, pp 42–51, 2003.
- [23] D. Liu, P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," *Proceedings of the 10th ACM Conference on Computer and Communication Security*, Washington DC, 2003.
- [24] M. Ramkumar, "Securing Ad Hoc Networks With "Asymmetric" Probabilistic Key Predistribution Schemes," *IEEE Information Assurance Workshop*, West Point, NY, 2006.