I-HARPS: An Efficient Key Pre-distribution Scheme

Mahalingam Ramkumar
Department of Computer Science and Engineering
Mississippi State University

Abstract

We introduce an efficient random key pre-distribution scheme (RKPS) whose performance is 2 to 3 *orders of magnitude* better than schemes of comparable complexity in the literature. This dramatic improvement is achieved by increasing *insecure* storage complexity (for example using external flash memory). The proposed scheme is a combination of the Kerberos-like key distribution scheme (KDS) proposed by Leighton and Micali, and random key pre-distribution schemes based on subset intersections.

We also investigate a simple security policy, DOWN (decrypt only when necessary) (which along with very reasonable assurances of tamper resistance / read-proofness could ensures that no more than *one* secret an be exposed by tampering with a node), and its effect on the security of key pre-distribution schemes. The proposed scheme lends itself well for efficient implementation of the DOWN policy, and therefore in practice could be a secure and efficient alternative to more complex conventional key distribution schemes.

1 Introduction

A key distribution scheme (KDS) is a mechanism of distributing secrets to each node in a system, such that any two nodes[1] can authenticate each other. The process of authentication typically involves discovery of a shared secret, while simultaneously providing verification of their claimed identities.

KDSs could be divided into two broad categories. For the first category, which includes most commonly used KDSs like Kerberos (or any KDS based on the Needham-Schroeder symmetric key protocol [1]) and PKI, the secrets provided to each node are *independent*. In other words, secrets of a node do not provide *any* information regarding the secrets of *other* nodes.

For the second category or *key pre-distribution* schemes (KPS), secrets distributed to each node are *not* independent - they are all *derived* from a set of secrets chosen by a *trusted authority* (TA) (who deploys the network). With KPSs, a group of "colluding" nodes could pool their secrets together to compromise the entire system (or obtain secrets of *all* nodes). There is thus a concept of n-secure KPSs. A n-secure KPS can resist collusions of up to n nodes.

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. A primary measure of complexity of a KPS is k - the *number of secrets* that need to be stored in each node. The *efficiency* of a KPS then, is usually expressed as a ratio of n vs k.

Since Blom et al [2] discovered that it is possible to trade-off security and complexity, many KPSs have been proposed in literature - the primary difference between the KPSs being the *nature of the trade-off employed*. In this paper, we introduce a novel KPS, I-HARPS (Id-HAshed Random Preloaded Subsets). The trade-off employed by the proposed scheme involves achieving significant reduction in the size of k (or size of *secure* storage) *and* computational complexity, by *increasing insecure* storage complexity.

The main motivation for the approach is the realization that *insecure storage complexity may not be a crucial issue* in many application scenarios. Flash based SD cards around the corner are expected have up to 8 GB of storage[2]. Thus for wireless devices like hand held communication equipment with add-on flash memory capabilities, employing a few megabytes, or even tens of megabytes of that storage for the KDS is not impractical.

By increasing insecure storage complexity, we show that a dramatic reduction of *secure* storage complexity (k), and computational complexity, is possible. Alternately, for the same secure storage complexity, a significant increase in security (or n - the number of colluding nodes that the KPS can resist) is possible. The proposed scheme is essentially a combination of a KDS proposed by Leighton[3] et al in Ref. [3], and the idea of key pre-distribution schemes involving random allocation of subsets of keys to each node, that have been investigated by many researchers in the literature [4] - [12].

The rest of this paper is organized as follows. In Section 2 we provide a brief introduction to KPSs, and the KDS by Leighton et al [3]. In Section 3 we introduce I-HARPS, and an analysis of its security. We also compare the performance of I-HARPS with other KPSs. We shall see that I-HARPS could be over *300 times more efficient* than other random KPSs of comparable complexity, and about 10 times more efficient than Blom's KPS (which however is computationally very expensive), with some "reasonable" provisioning for *additional insecure storage*.

KPSs are typically expected to be employed in "trusted" devices with some hardware protection of stored secrets. In Section 4 we review problems and solutions involving protecting

[1]More generally, any group of nodes.

[2]Even though SD stands for "secure" digital we do not intend to use the security mechanism offered by SD cards for protecting information stored in the SD card. Other flash memory cards like CF (compact flash), SM (smart media), MS (memory stick), MMC (multimedia card) and xD-picture cards offer comparable storage at comparable prices.

[3]The authors refer to this scheme as "a software based approach."

secrets in trusted devices and indicate how a recently proposed security policy - DOWN (decrypt only when necessary) [13] can substantially improve the ability of trusted devices to protect their secrets. In Section 5 we compare different KPSs in conjunction with the DOWN policy - which provides an assurance that *not more than one secret* can be exposed by tampering with a device. When I-HARPS (of very reasonable complexity) is used in conjunction with DOWN, we show that an attacker may have to compromise secrets from many *tens of millions* of nodes in order to make any kind of dent in the security of the KPS.

2 Key Pre-distribution

A KPS consists of a trusted authority (TA), and N nodes with unique IDs (say $ID_1 \cdots ID_N$). The TA chooses P secrets $\mathcal{R}$ and two operators $f()$ and $g()$. The operator $f()$, is used to determine the secrets $\mathcal{S}_i$ that are preloaded in node i. Any two nodes i and j, with preloaded secrets $\mathcal{S}_i$ and $\mathcal{S}_j$ can discover a unique shared secret K_{ij} using a *public* operator $g()$ without further involvement of the TA. The restrictions on $f()$ and $g()$ in order to satisfy these requirements can be mathematically stated as follows:

$$\begin{aligned} \mathcal{S}_i &= f(\mathcal{R}, ID_i); \\ K_{ij} &= g(\mathcal{S}_i, ID_j) = g(\mathcal{S}_j, ID_i) \\ &= f(\mathcal{R}, ID_j, ID_i) = f(\mathcal{R}, ID_i, ID_j). \end{aligned} \tag{1}$$

As $g()$ is public, it possible for two nodes, just by exchanging their IDs, to execute $g()$ and discover a unique shared secret. As the shared secret is a function of their IDs, their ability to arrive at the shared secret provides mutual assurances to i and j that the other node possesses the necessary secrets $\mathcal{S}_j$ and $\mathcal{S}_i$, respectively. The secrets preloaded in each node is referred to as the node's *key-ring*. We shall represent by k, the size of the key ring.

2.1 Why KPS?

The main disadvantage of Kerberos-like approaches [1] is the need for an online server for mediation of interaction between nodes, which is not satisfactory under scenarios where *ad hoc* mutual authentication is necessary. While PKI does not have this issue, there are three major issues that render PKI unsuitable for many application scenarios:

1. Large computational and bandwidth overheads due to the need for asymmetric cryptography.
2. Efficient dissemination of public keys in very large-scale deployments may not be possible as certificate chains [14] needed for mutual authentication could become very long.
3. For many applications[4], the ability to escrow keys may actually be desirable.

[4]For example, smart cards with autonomous computational capabilities. As such devices would be expected to *self-destruct* on sensing tampering attempts, false-alarms may result in locking away valuable encrypted data from genuine users, unless key escrow is possible.

KPSs cater for ad hoc mutual authentication by just exchanging IDs. Further, the problem of *choosing* public IDs (for efficient dissemination of public keys) is not an issue with KPS. Unlike asymmetric schemes where it is not possible in general to choose public keys (the choice of the private keys determine the choice of the public keys), with KPSs one can always choose their public ID - say by hashing a descriptive string like ["First-Name LastName Affiliation"]). In other words, for any KPS, the private keys (the k secrets assigned to each node) are determined by the "public keys" (public ID). Also, KPSs by their very nature, cater for key escrow.

While the problems associated with efficient public key dissemination and key escrow are simultaneously overcome by using identity based cryptography (IBE) [15], they impose the need for *ongoing involvement* of a trusted third party (as in Kerberos-like models).

2.2 Deterministic KPS

KPSs themselves may be divided into two broad categories - deterministic and random KPS. Most KPSs based on finite field arithmetic [2], [16] - [18] belong to the former category. For example, in a n-secure Blom's scheme, the TA chooses $\binom{n+1}{2}$ secrets in $\mathbb{Z}_P = \{0, 1, \ldots, P-1\}$ (where P is a large enough prime), and generates a polynomial

$$f(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} a_{ij} x^i y^j \bmod P, \;\; x, y, a_{ij} \in \mathbb{Z}_P. \tag{2}$$

where $a_{ij} = a_{ji}$ are $\binom{n+1}{2}$ independent secrets chosen by the TA. Every node is assigned a *unique* public ID[5] from $\mathbb{Z}_P$. A node A (node with public ID $A \in \mathbb{Z}_P$) receives $g_A(x) = f(x, A)$ securely ($g_A(x)$ has $n+1$ coefficients, corresponding to $k = n+1$ secrets of the node A) from the TA. Two nodes A and B can calculate K_{AB} = K_{BA} = $f(A, B) = f(B, A)$ = $g_A(B)$ = $g_B(A)$ independently.

A n-secure deterministic KPS is unconditionally secure as long as n or less nodes have been compromised. If more than n nodes are compromised however, the *entire* KPS is compromised - or failure of the KPS occurs catastrophically. The most efficient of deterministic KPSs thus far (Blom's scheme) requires only $k = n+1$ keys in each node to be n-secure. However, Blom's scheme is computationally expensive.

To overcome the two major limitations of KPSs based on finite-field arithmetic (computational complexity and catastrophic failure) Gong et al [19] and Mitchell et al [20] investigated KPSs based on allocation of a subset of keys to each node from a pool of keys. The shared secret between any two nodes is then derived from *all* shared keys (say a one way function of all shared keys). While very naive approaches were used by Gong et al [19] for *allocation* of keys, Mitchell et al [20] were influenced by the seminal work of Erdos et al [21] on subset intersections. However the complexity involved in allocation of keys in such approaches, also makes it difficult for the nodes to *determine* the keys they share[6] in order to establish a shared secret.

[5]The size of P limits the possible network size.

[6]To overcome this nodes might have to exchange P-bit messages to indicate

2.3 Random KPSs

Dyer et al [4] were the first to point out that *random allocation of subsets* (instead of complex deterministic strategies) works "reasonably well." More recently, this idea has been employed by various researchers [5] - [10] in the context of sensor networks, and [11] - [12] for mobile ad hoc networks. In this paper, we refer to all KPSs based on random allocation of subsets as RPS (**r**andom **p**reloaded **s**ubsets).

Leighton et [3] (in the same paper in which they proposed the simple and elegant KDS which will be discussed in the next section) also proposed the first random key pre-distribution scheme[7](which we shall refer to as LM-KPS). Further unlike RKPSs that followed, LM-KPS is *not based on subset intersections*. LM-KPS is based on distributing keys with different "hash depths" to each node.

HARPS [12], perhaps the most efficient random KPS thus far[8], is a generalization of RPS and LM-KPS.

Formally, a (P, k) RPS employs a TA who chooses an indexed set of P keys $K_1 \cdots K_P$. Each node has a unique ID. The TA chooses public random function $F_{RPS}()$, which when "seeded" by a node ID, yields the allocation of keys for the node. Thus for a node A (node with unique ID A)

$$\begin{aligned} F_{RPS}(A) &= \{A_1, A_2, \ldots, A_k\}, \\ \mathbb{A} &= \{K_{A_1}, \ldots, K_{A_k}\}. \end{aligned} \qquad (3)$$

where $1 \leq A_i \leq P, A_i \neq A_j$ for $i \neq j$. In other words $F_{RPS}()$ generates a *partial* random permutation of $\{1 \cdots P\}$. The k-length sequence $\{A_1, A_2, \ldots, A_k\}$ is the index of the keys preloaded in node A (or node with ID A). $\mathbb{A}$ is the set of secrets preloaded in A. Note that the indexes are public (as the node ID and $F_{RPS}()$ are public).

In the (k, L) LM scheme, the TA chooses an indexed set of k secrets $K_1 \cdots K_k$, a cryptographic hash function $h()$, and a public random function $F_{LM}()$. For a node A,

$$\begin{aligned} F_{LM}(A) &= \{a_1, a_2, \ldots, a_k\}, 1 \leq a_i \leq L \forall i. \\ \mathbb{A} &= \{{}^{a_1}K_1, {}^{a_2}K_2, \ldots, {}^{a_k}K_k\}. \end{aligned} \qquad (4)$$

Or $F_{LM}()$ generates a k-sequence of uniformly distributed random integer values between 1 and L. The node A is preloaded with k keys. The i^{th} preloaded key is node A is derived by repeatedly hashing i^{th} TAs key a_i times. The parameter L is the maximum hash depth. The notation ${}^{i}K_j$ represents the result of *repeatedly* hashing of K_j, i times, using a (public) cryptographic hash function $h()$.

In (P, k, L) HARPS, the TA chooses P keys $K_1 \cdots K_P$, and each node is loaded with a *hashed* subset of k keys. The TA has an indexed set of P secrets, a cryptographic hash function $h()$ and a public random function $F_{HARPS}()$. For a node A,

$$\begin{aligned} F_{HARPS}(A) &= \{(A_1, a_1), (A_2, a_2), \ldots, (A_k, a_k)\}, \\ \mathbb{A} &= \{{}^{a_1}K_{A_1}, {}^{a_2}K_{A_2}, \ldots, {}^{a_k}K_{A_k}\}. \end{aligned} \qquad (5)$$

The first coordinate $\{A_1, A_2, \ldots, A_k\}$ represents the index of the keys chosen to be preloaded in node A, and the second coordinate $\{a_1, a_2, \ldots, a_k\}$, the number of times each chosen key is hashed (using cryptographic hash function $h()$) before they are preloaded in the node A.

Note that LM-KPS and RPS are actually special cases of HARPS. LM-KPS is HARPS with $P = k$, and RPS is HARPS with $L = 0$ (keys are not hashed before pre-loading) or $L = 1$ (all keys are hashed once before preloading)[9].

The concept of n-secureness is however, not an adequate description of an RKPS. For any RKPS, an attacker, by exposing secrets from n_e nodes could discover shared secrets between arbitrary nodes with a some probability p_e. Thus a more appropriate description of a RKPS could be (n_e, p_e)-secure KPS.

In general RKPSs are less efficient than the schemes based on finite field arithmetic - even though it is still $k = O(n)$ (except for the LM-KPS scheme which needs $k \approx O(n^3)$). For instance, to achieve (n_e, p_e)-security, HARPS [12] needs $k \approx n_e \sqrt{\mathrm{e}} \log\left(\frac{1}{p_e}\right)$ keys, and schemes based on random preloaded subsets [4] - [11] require $k \approx n_e \mathrm{e} \log\left(\frac{1}{p_e}\right)$ (or HARPS is more efficient by a factor $\sqrt{\mathrm{e}}$). RKPSs are also *computationally inexpensive* - they need only pure symmetric cryptography primitives like hash functions and / or block ciphers (multiplication is not needed unlike in Blom's scheme).

Further, with a (n_e, p_e)-secure RKPS, exposing keys from n_e devices enables an attacker *only* to determine *shared secrets between* nodes[10]. To actually determine *all* secrets in some node by exposing secrets from other nodes, the attacker may have to expose keys from a *significantly higher* number of nodes. We refer to this type of attack - aimed at exposing *all* secrets from a node by exposing secrets from *other* nodes - as a *synthesis* attack[11]. Thus an even more appropriate characterization of an RKPS would be as $\{(n_e, p_e), (n_s, p_s)\}$-secure, where an attacker needs to compromise secrets from

1. n_e nodes to discover shared secrets (between nodes other than the compromised nodes) with a probability p_e, and
2. n_s nodes to accomplish a *synthesis* attack with a probability p_s.

In general, for $p_e = p_s$ $n_s >> n_e$. For RPS and LM-KPS n_s is an order of magnitude higher than n_e. For HARPS, n_s is more than two orders of magnitude higher than n_e [12].

the indexes of the keys they share. However, this approach does not implicitly provide authentication of the node IDs.

[7]The authors refer to this scheme as "the second basic scheme" in [3].

[8]Schemes proposed in [9] and [10] - which combine Blom's scheme with RPS schemes could be a little more efficient than HARPS. However, the performance of such schemes designed for a particular n, deteriorates very rapidly for $n' > n$ [12].

[9]In practice choosing $L = 1$ instead of $L = 0$ does not have any implication on the security of shared secrets between nodes. The only advantage of choosing $L = 1$ is that compromise of keys in nodes does not result in compromise of TA's keys.

[10]With which an attacker can impersonate a node for purposes of "fooling" other nodes. More specifically it is *not* possible to fool the TA.

[11]A synthesis attack enables an attacker to even fool the TA.

2.4 Leighton - Micali KDS

The LM-KDS [3] (not to be confused with LM-KPS discussed earlier) is based on a master key, and a strong cryptographic hash function $h()$. The scheme consists of a trusted authority and a set of N nodes $\mathfrak{M}$. The trusted authority chooses a master key K. Node $i \in \mathfrak{M}$ is provided with the secret $K_i = h(K, i)$. For sending a message to node $j \in \mathfrak{M}$ (which has secret $K_j = h(K, j)$), node i performs a look up in a *public* repository (created by the TA, with N^2 entries) for P_{ij} (P_{ij} is not a secret), where $P_{ij} = h(K_j, i) \oplus h(K_i, j)$, and calculates K_{ij} as $K_{ij} = P_{ij} \oplus h(K_i, j) = h(K_j, i)$. The message M to be sent to j may be encrypted with a random session key K_S, and sent to j as

$$M_{ij} = [i \parallel E_{K_{ij}}(K_S) \parallel E_{K_S}(M)]. \qquad (6)$$

Node j can easily calculate $K_{ij} = h(K_j, i)$ as it has access to K_j.

However, in practice, it may not be feasible to maintain a public repository with N^2 keys. So the TA may actually need to be on-line to calculate P_{ij} and provide it to the nodes "on demand". However, once a node i obtains P_{ij} (to communicate with node j), it does not have to get P_{ij} again. Further, any node i may not need to know P_{ij}s for *all* possible $j \in \mathfrak{M}$. So the node i could just store some P_{ij}s for some nodes (even in some insecure storage location for easy access).

However, P_{ij}s stored in insecure locations, could have been modified - and therefore need to be authenticated. For this purpose, a second authentication key is used. The TA chooses an additional master key ${}^aK$, and provides node i with ${}^aK_i = h({}^aK, i)$. Additionally, one more public value A_{ij} is used to authenticate each P_{ij}, where $A_{ij} = h({}^aK_i, h(K_j, i)) = h({}^aK_i, K_{ij})$. As the node i has ${}^aK_i$, it can check if K_{ij} obtained from a P_{ij} (provided by an untrusted source) is valid.

The main difference between the LM-KDS and schemes (like Kerberos) based on the symmetric Needham-Schroeder protocol [1] (which also require a trusted on-line server), is that the information P_{ij} that the node gets from the server *need not be secret* (nodes do not even need to authenticate themselves to the server to receive P_{ij}s). Further, the TA is not required to be on-line for *every* communication attempt between i and j - nodes need to access the TA *only once*. It is also possible for node i to get P_{ij}s for a large number of js that node i may desire to communicate with in the future, in a single attempt.

The security of LM-KDS rests on the assumption that the master key cannot be compromised (while for Kerberos-like models the assumption is that the trusted server cannot be compromised)[12]. The LM-KDS could however easily be extended to using multiple master keys (say t such systems used together, with master keys $K^1 \cdots K^t$). In this case an attacker would have to compromise the master keys from *all* t systems to break the system. The authentication secret K_{ij} in this case would be $K_{ij} = K^1_{ij} \oplus K^2_{ij} \oplus \cdots \oplus K^t_{ij}$.

[12]As it may be easier to unconditionally protect a single key in a device rather than multiple keys, it could be argued that the LM-scheme is more secure than Kerberos-like models (if *cryptanalytic* attacks are considered impractical).

The primary *disadvantage* of LM-KDS is that it does not provide a good solution for *revocation* of nodes - some alternate mechanism needs to be used for this purpose. For Kerberos like models this is not an issue as the TA would just refuse to honor requests from revoked nodes for authenticating itself to other nodes.

2.4.1 Basic KDS vs LM-KDS

In the *basic* KDS, for a system consisting of N nodes, the TA chooses $\binom{N}{2}$ secrets (one for each pair) and assigns each node with $N - 1$ secrets. After the keys are assigned in each node, there is no need for the involvement of the TA for mutual authentication of nodes. Thus the basic KDS is a KPS scheme. In fact a very secure KPS scheme - no matter how many nodes are compromised, nodes that *not* compromised are not affected.

The LM-KDS can also be used to facilitate authentication of node *without the involvement of the TA* (or work like a KPS scheme). In this case, each node (say node i) just needs to store $N - 1$ P_{ij} values. While both approaches (basic KDS vs LM-KDS) have the same storage complexity, there is one noteworthy difference. For the basic KDS, $N - 1$ *secrets* need to be stored. For LM-KDS the $N - 1$ P_{ij} values need not be protected.

2.5 The Perfect KPS?

We can already see that if *insecure storage complexity is not an issue* the LM-KDS is indeed a very efficient KPS! Each node just needs to store one secret[13]! Further, no coalition of nodes can compromise secrets of other nodes (as long as cryptanalytic attacks are infeasible). In fact for medium scale deployments involving say few tens of millions of nodes, LM KDS may be a feasible solution (even if the network size is 64 million, 1 GB of storage would be sufficient - which is perhaps not totally impractical).

However, in order for a deployment to be highly scalable, and to fully utilize the advantages KPSs offer over other key distribution schemes, it may be necessary to choose a much larger "ID space." For instance if we desire to assign public IDs based on a one-way function of "FirstName LastName Affiliation" the ID space should at least be 128 bits long to be useful (to ensure that collitions are highly improbable).

Under such a condition, it is still possible for nodes to store the P_{ij} values for each deployed node (which may be a few tens of millions) However, it is not possible to predict the IDs of the nodes that "join the network" *after* the deployment. Thus whenever new nodes join the network every node should be provided with the corresponding P_{ij}s - which may not be practical[14].

In the next section we introduce a novel KPS scheme, I-HARPS, which overcomes this problem. With "more reasonable" requirement of insecure storage of a few megabytes or tens of megabytes, I-HARPS allows for very high scalability (practically without bounds).

[13]We shall ignore the authentication key ${}_aK$ for the moment.

[14]Though this is much more practical than providing each node with an additional *secret* - which would be required for the basic KDS.

3 I-HARPS

I-HARPS (like HARPS[15]) is also determined by three parameters - P - the number of secrets the TA chooses, k - the number of secrets in each node, and L. However, unlike HARPS where L is the maximum "hash depth," in I-HARPS, L determines the additional *insecure storage complexity*. More specifically, the insecure storage complexity is $k(L-1)$.

The TA chooses P secrets $\{{}^1K \cdots {}^PK\}$ (which we shall see are actually the master keys of P independent LM-KDS), a cryptographically strong hash function $h()$ and a *public* function $F()$. Let N represent the total number of nodes in the system. Each node has a unique ID. Like most KPSs, the network size is only limited by the number of bits chosen to represent the ID[16].

A node with ID A gets a set of k secrets $\mathbb{A}$, and $k(L-1)$ values $\mathfrak{A}$ which are determined by the two one way functions $h()$ and $F()$ as follows:

$$F(A) = \{(A_1, a_1) \cdots (A_k, a_k)\}, \tag{7}$$

for $1 \le A_i \le P, A_i \neq A_j \forall i \neq j, 1 \le a_i \le L$, and

$$\begin{aligned} \mathbb{A} &= \{{}^{A_1}K_{a_1} \cdots {}^{A_k}K_{a_k}\}, \text{ where} \\ {}^iK_j &= h({}^iK, j), \end{aligned} \tag{8}$$

and

$$\mathfrak{A} = \left\{ \begin{array}{c} {}^{A_1}P_{a_1,1} \cdots {}^{A_1}P_{a_1,a_1-1} \; {}^{A_1}P_{a_1,a_1+1} \cdots {}^{A_1}P_{a_1,L} \\ {}^{A_2}P_{a_2,1} \cdots {}^{A_2}P_{a_2,a_2-1} \; {}^{A_1}P_{a_2,a_2+1} \cdots {}^{A_2}P_{a_2,L} \\ \vdots \\ {}^{A_k}P_{a_k,1} \cdots {}^{A_k}P_{a_k,a_k-1} \; {}^{A_k}P_{a_k,a_k+1} \cdots {}^{A_k}P_{a_k,L} \end{array} \right\} \tag{9}$$

where

$${}^{A_i}P_{a_i,j} = h({}^{A_i}K_{a_i}, j) \oplus h({}^{A_i}K_j, a_i). \tag{10}$$

In other words, the TA chooses P LM-KDS master keys. Each system however is limited to L "users" (or only L^2 P_{ij} values need to be generated by the TA). Node A is provided with keys from k of the P systems. The specific choice of k out of P systems for node A is determined by $A_1 \cdots A_k$ (through the public function $F(A)$). In each of the k systems, the ID of A is given by $1 \le a_i \le L$ (ID a_i in system A_i). Thus each node needs to store $L-1$ values for each of the k systems - introducing an insecure storage requirement of $k(L-1)$.

In order to establish a shared secret with B, node A needs to evaluate $F(A)$ and $F(B)$ to determine the shared indexes of the LM-KDS instances. Let us assume nodes A and B share m systems with master keys ${}^{S_1}K \cdots {}^{S_m}K$, and the ID (between 1 and L) of A and B in the m systems are $(a_1 \cdots a_m)$ and $(b_1 \cdots b_m)$ respectively. The shared secret between A and B, or K_{AB} is then

$$K_{AB} = h({}^{S_1}K_{b_1}, a_1) \oplus h({}^{S_2}K_{b_2}, a_2) \oplus \cdots \oplus h({}^{S_m}K_{b_m}, a_m)$$

[15]HARPS employs a combination RPS and LM-KPS. I-HARPS employs a combination of RPS and LM-KDS.

[16]In Blom's KPS [2] it is limited to P - the number of elements in the finite field $\mathbb{Z}_P$ over which the polynomials are evaluated.

Note that as in LM-KDS, node B can easily calculate $h({}^{S_i}K_{b_i}, a_i)$ as it has the necessary secrets ${}^{S_i}K_{b_i}, 1 \le i \le m$. Node A can obtain each $h({}^{S_i}K_{b_i}, a_i)$ as

$$h({}^{S_i}K_{b_i}, a_i) = {}^{S_i}P_{a_i,b_i} \oplus h({}^{S_i}K_{a_i}, b_i), \tag{11}$$

by looking up the P_{ij} values from $\mathfrak{A}$ in insecure storage.

3.1 Analysis of I-HARPS

Let

$$\xi = \frac{k}{P}. \tag{12}$$

Now consider the i^{th}LM-KDS system where $1 \le i \le P$. In order for two nodes (say A and B) to utilize the i^{th}KDS, both should *have* a secret for the i^{th}KDS - which occurs with a probability ξ^2 (or on an average two nodes share $P\xi^2 = k\xi$ systems).

Let us assume that an attacker has compromised *all* secrets from n nodes. In order to compromise the secret shared between A and B provided by system i (where A has an ID $1 \le a \le L$ and B has ID $1 \le b \le L$ in the i^{th}system), the attacker needs to find, in his ill-gotten collection of exposed secrets, *either* ${}^iK_a$ or ${}^iK_b$. The probability that the attacker finds exactly u instances of system i keys in n nodes is

$$B_\xi(n,u) = \binom{n}{u}\xi^u(1-\xi)^{n-u}, \tag{13}$$

and the probability that u instances of the keys (of system i) correspond to either a or b is $\frac{2}{L}$. Thus the probability that two nodes can use the i^{th}KDS safely is

$$\varepsilon = \xi^2 \sum_{u=0}^{n} B_\xi(n,u)\left(1 - \frac{2}{L}\right)^u. \tag{14}$$

In order to compromise the shared secret the attackers have to determine all elementary secrets which make up the final shared secret K_{AB}. Thus the probability p_e that an attacker who has compromised n nodes can compromise *shared secrets* of arbitrary nodes is

$$p_e = (1-\varepsilon)^P. \tag{15}$$

By similar reasoning the probability that an attacker can compromise *all* secrets of a node by exposing secrets from other nodes is

$$p_s = (1-\varepsilon_s)^P, \tag{16}$$

where

$$\varepsilon_s = \xi \sum_{u=0}^{n} B_\xi(n,u)\left(1 - \frac{1}{L}\right)^u. \tag{17}$$

Note that there are two differences between Eqs (14) and (17). The first is the difference in the first term (ξ instead of ξ^2) as each node has $k = P\xi$ keys - while only $P\xi^2$ keys are *shared* between nodes. The second difference is in the last term which has $\frac{1}{L}$

instead of $\frac{2}{L}$. In the former case it is enough for the attacker to determine *either* the key of A *or* B for a system - which is not the case in the latter.

We have however, in Eq(14), ignored the fact that it is possible that the IDs of the two nodes (whose shared secret the attacker is trying to compromise) may have the *same ID* in the i^{th}system. Under this condition (the probability of which is $\frac{1}{L^2}$), the attacker's job is a little more difficult (he has to have that *particular* key instead of *one of two* keys). Thus Eq (14), is more accurately written as

$$\varepsilon = \xi^2 \sum_{u=0}^{n} B_\xi(n,u) \left(\frac{(L^2-1)(1-\frac{2}{L}) + (1-\frac{1}{L})}{L^2} \right)^u . \quad (18)$$

3.2 Comparison With Other KPSs

Figure 1 (left) provides a comparison of 5 KPSs in terms of the probability with which an attacker who has compromised n nodes, can discover shared secrets of other nodes (for the Y-axis we use $-\log_{10} p_e$ - so higher the better). For the sake of comparison, all KPSs have the same $k = 1000$. For HARPS we have chosen $P = 15000, L = 1024$. For I-HARPS $P = 15000, L = 1024$, for RPS $P = 20000$, for LM-KPS $P = k = 1000, L = 1024$, and $k = 1000$ for Blom's scheme (or 999-secure Blom's scheme). While the other KPSs do not require any additional insecure storage for I-HARPS, if $k = 1000$, $L = 1024$ and each key is of length 128-bits (16 bytes), the P_{ij}s (which are also the same length as the keys) would require $16k(L-1)$ bytes or less than 16 MB of storage.

Note that even for deterministic KPSs like Blom's scheme, there is always a probability that the attacker can "guess" the shared secret between two nodes. For instance, if the final shared secret is a 128-bit secret, the probability that the attacker can determine the secret[17] is $p_e = \frac{1}{2^{128}} \approx 3 \times 10^{-39}$. For Blom's scheme the "probability of compromise" is therefore fixed at roughly 10^{-39} for $n \leq 999$. However for $n \geq 1000$ the probability of compromise is unity (or $log_{10} p_e = 0$).

Note that I-HARPS is very much usable even when 8000 nodes have been compromised! Beyond $n = 8000$ (not shown in the plots), the probability of compromise is about

1. 1 in a billion ($p_e \approx 10^{-9}$) when 9,000 nodes have been compromised,
2. 1 in a million when 12,000 nodes have been compromised,
3. 1 in a thousand when 17,500 nodes are compromised.
4. and $p_e = 0.5$ for $n = 27,500$.

The trade-off is of course the need for additional insecure storage - 1024×1000 P_{ij} values have to be stored in each node. For example, if all keys (and hence the P_{ij}s) are 128-bits, an additional 16 Megabytes of storage is required for I-HARPS.

The three lines in the lower left corner of Fig 1 (left) correspond to HARPS, RPS and LM-KPS respectively[18]. A zoomed view of the three lines are shown in Fig 1 (right).

[17]Which also goes to show that as long as the probability of compromise is low, an RKPS is in no way inferior to a deterministic KPS.

[18]Which are literally "out of the chart"!

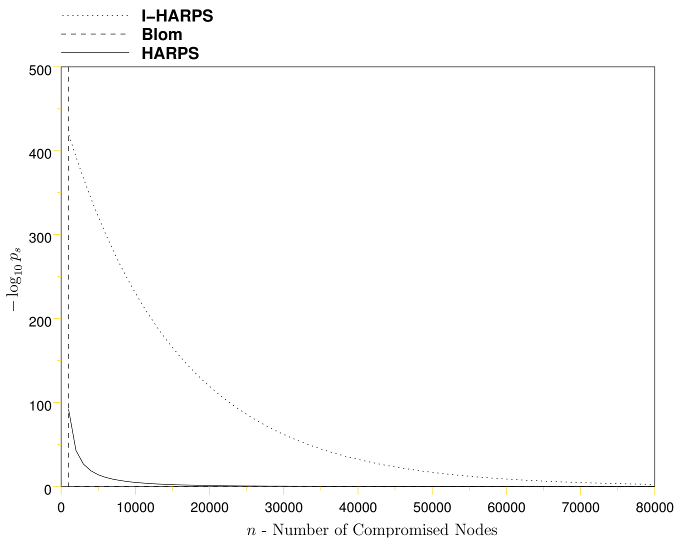


Figure 2: Resistance of KPSs (Blom's scheme, HARPS and I-HARPS) to node synthesis.

Figure 2 depicts the probability of *node synthesis* p_s - or the probability with which an attacker can compromise *all* secrets from a node by exposing keys from other nodes. The figure has plots ($\log_{10} p_s$ vs n) for I-HARPS, HARPS and Blom's scheme. For Blom's scheme, $p_e = p_s = 1$ for $n = n_e = n_s \geq k = 1000$. However RKPSs in general deteriorate more gracefully. With $n = 9000$ compromised nodes, the attacker can expose all keys from roughly *one in a million* nodes for HARPS. For I-HARPS, the probability p_s associated with different number of nodes the attacker compromises is

1. one in a trillion ($p_s = 10^{-12}$) - $54,000$ nodes
2. one in a billion - 59000 nodes,
3. one in a million - $66,000$ nodes (9000 for HARPS).

3.2.1 Effect of L (Insecure Storage Complexity)

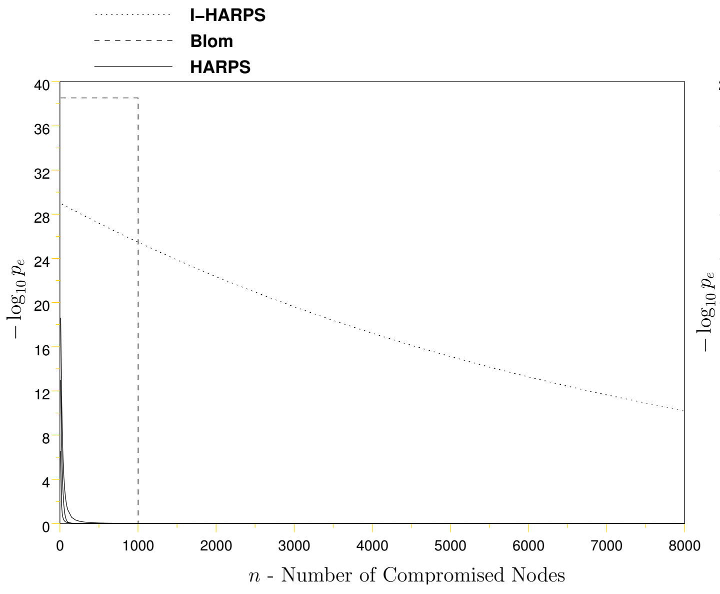
Figure 3 depicts the effect of L on the performance of of I-HARPS. Obviously, as L increases we expect the I-HARPS to improve substantially. After all, we already know that if there is no limit to L we could just use the LM-KDS with N (network size) stored P_{ij} values which is secure against collusion of an unlimited number of nodes!

With more practical restrictions on L, it can be seen that if L is increased (and P, k are unchanged), n can be increased by the same factor to keep p_e constant, by considering a first order approximation of Eq (14),

$$\varepsilon = \xi^2 \sum_{u=0}^{n} B_\xi(n,u) \left(1 - \frac{2}{L}\right)^u \approx \xi^2 \left(1 - \frac{2}{L}\right)^{\xi n}, \quad (19)$$

and the fact that

$$(1-x)^y \approx (1-x/2)^{2y} \approx 1 - xy \text{ for } x << 1. \quad (20)$$



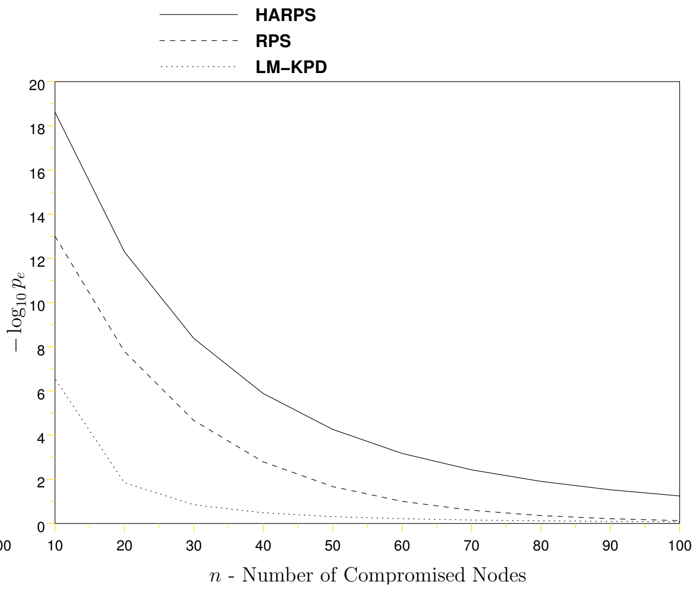


Figure 1: Left: Comparison of 5 KPSs with the same value of $k = 1000$, in terms of probability of compromise of shared secrets vs number of compromised nodes. Right: Zoomed in version of the lower left corner of the figure in the left to show performance of HARPS, RPS and LM-KPS.

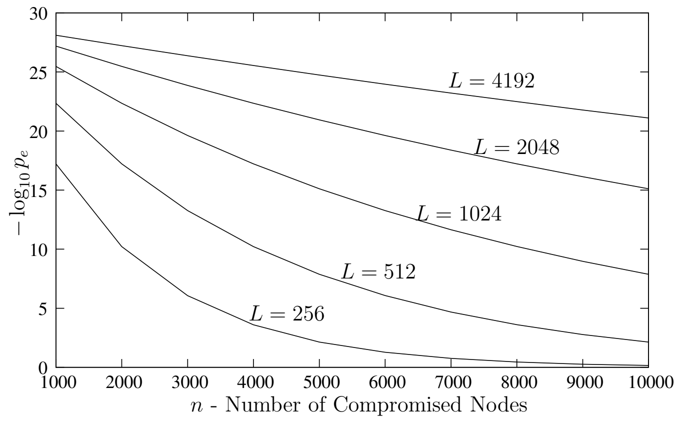


Figure 3: Performance of I-HARPS with $P = 15000, k = 1000$ for different values of L. Larger L implies larger insecure storage complexity.

3.2.2 Authentication Key

So far we have ignored the additional authentication key used by the LM-KDS. The main need for the additional authentication key is to ensure that modifications to P_{ij}s stored in insecure locations could be detected. Thus in practice the authentication key is not needed if

1. the P_{ij} values are from a trusted source, and provided in read-only storage device, or
2. some key based hash function (say using the user secret ${}^{i}K_A$ for the i^{th}LM-KDS) is used for authenticating all ${}^{i}P_{Aj}$ values for $1 \leq j \leq L$.

3.2.3 Hashing vs Storage Trade-offs

RKPSs exploit two fundamental "dimensions" - one provided by *uniqueness of intersections of large subsets* consisting of *independent* keys, and the other provided by *generating many keys* from each independent key. RPS schemes [4] - [11] make use of the former. LM-KPS [3] makes use of the latter. HARPS and I-HARPS use both. Specifically, HARPS achieves this without requiring extra storage, while I-HARPS calls for extra storage. It is very easy to see that a (P, k, L) I-HARPS (as does (P, k, L) HARPS) reduces to (P, k) RPS when $L = 1$ (no storage needed for I-HARPS, and no hashing for HARPS)

It is interesting therefore, to see how much storage, can be "emulated" by hashing. Figure 4 depicts plots comparing HARPS with $L = 1024$ with I-HARPS (with the same P and k as HARPS) for $L = 3$ and $L = 4$. As HARPS with $L = 1024$ falls "between" the cases of I-HARPS with $L = 3$ and $L = 4$, hashing in HARPS could be said to "emulate" a storage complexity between $2k$ and $3k$ (only $L - 1$ public values need to be stored for each secret).

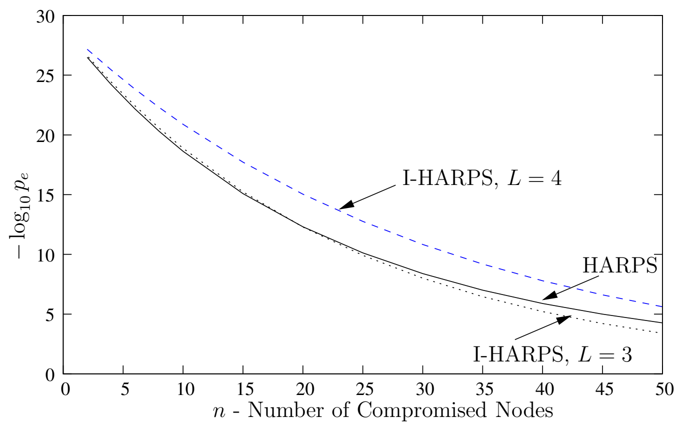


Figure 4: Comparison of HARPS ($L = 1024$) with I-HARPS ($3 \leq L \leq 4$). Hashing in HARPS "emulates" a storage complexity between $3 - 1 = 2$ and $4 - 1 = 3$. For both KPSs $P = 15000, k = 1000$.

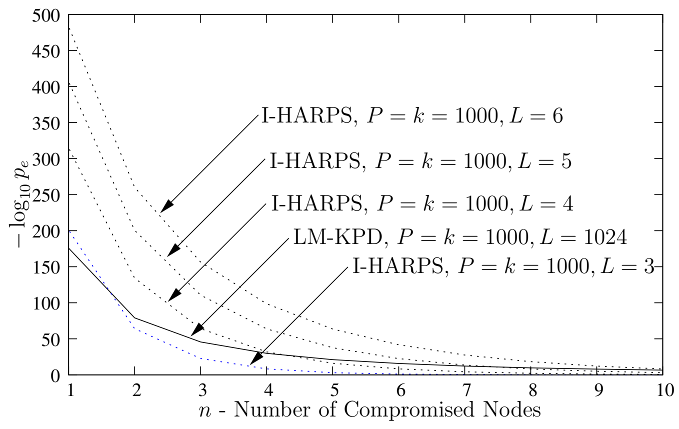


Figure 5: Comparison of LM-KPS ($L = 1024$) with I-HARPS (small $3 \leq L \leq 6$). For both KPSs $P = k = 1000$.

Figure 5 is a comparison of LM-KPS with a special case of I-HARPS with $P = k$. For LM-KPS $L = 1024$, and for I-HARPS we have plots for $P = k = 1000$ with $L = 3 \cdots 6$. In both cases (HARPS vs I-HARPS in Figure 4 and LM-KPS vs I-HARPS in Figure 5) note that the storage emulation offered by hashing increases as n increases. In other words, employing hashing instead of storage, while less efficient, deteriorates at a lower rate. It is not surprising that the LM scheme exploits the "second dimension" (the only dimension it exploits) better than HARPS - as HARPS can "exploit this dimension" only for *shared* keys. With LM all k keys are shared (as $P = k$), while with HARPS only ξk keys are shared on an average.

That HARPS deteriorates more gracefully than I-HARPS is also readily apparent by considering that a 300-fold difference between HARPS and I-HARPS in terms of complexity of attacks for discovering *shared secrets*, reduces to a mere 7-fold difference between the two in terms of complexity of synthesis attack!

4 DOWN with Trusted Computers

Key pre-distribution schemes are more likely to be used in devices with some assurance of tamper resistance - for example, "trusted computers" with some ability to "protect" their secrets. In this section we investigate a simple security policy - DOWN (decrypt only when necessary) [13] and its ability to enhance the ability of trusted devices to protect its secrets. In particular we show that some KPSs - especially RKPSs are more "DOWN friendly." We then proceed to analyze the effect of the DOWN policy on HARPS, I-HARPS and Blom's KPS.

4.1 Protecting Secrets in Trusted Computers

Trusted computers [25] are expected to possess unflinching *morals* (the *morality* of the computers would be dictated by the software that runs on the computer). They will not, in general, trust the owners or controllers of devices (the human operators). They cannot be directed to do something that violates the rules they are trusted to obey. Just as human beings are trusted not to disclose their secrets in client-server applications, trusted computers are trusted not to disclose *their* secrets.

Therefore, practical realization of trusted devices calls for two fundamental assurances from technology - tamper-resistance and read-proofing. However both properties are *not* independent [26]. For instance, with the ability to tamper with the software, an attacker could direct the trusted computer to "spit-out" its secrets. At the same time, the protected secrets could be used for authenticating the software (say using key based message authentication codes) - and thus prevent attackers from tampering with the software.

Trusted devices (for example a single chip computer in a smart card) can be at two fundamental states - *in-use* or *at-rest*. Obviously, the secrets have to be protected during both states.

4.1.1 In-Use State

While the computer is in-use, it is possible to use many sensors that actively monitor for intrusions, and delete secrets under suspicion of tampering attempts. Attackers could monitor electromagnetic radiations from chips for gaining clues about the secrets stored inside. But this could be prevented by employing proper shielding. Some attacks are also based on inducing faults in memory [28], [29] and employing differential power analysis [30]. Protection mechanisms against such attacks have also been investigated [31].

Another approach for attackers is to the use of sophisticated focused ion beam (FIB) techniques [32] to drill fine holes and establish a connection with the computer buses and constantly monitor the bits that traverse through the buses. In Ref. [33] the manufacturers claim protection against FIB attacks by employing an "active shield" consisting of a thin parallel grid of signal lines, covering the entire surface of the chip. Even if one line is damaged by the FIB, this would trigger appropriate counter measures (for example deleting all secrets). Perhaps even multiple layers of active shield could be used. As long as even *some* parts of the silicon (or some lines on the substrate) are protected by not providing a clear line of sight access (after all, it may not be possible to *bend* beams), it may be possible to build *private* circuits [34] to ensure that the attacker gains no knowledge (by tapping a few lines).

4.1.2 Protecting Secrets At-Rest

While rest encryption [35], is commonly used to protect databases at rest, the problem of rest encryption for trusted devices is very different. For the former, the secret used for encrypting the contents of the database is typically stored *outside* the database. For the latter, this is not possible as trusted computers *will not trust anyone else* with their secrets. So, in trusted computers, even if the secrets are encrypted for rest, the secret used for encrypting the secrets should be stored in some non-volatile memory (NVM) *inside* the trusted computer, and should be protected. This calls for a continuous source of power supply to the devices *even when they are at rest*, to monitor for potential intrusions and erase the secrets when an intrusion is sensed. However, it is not necessary that *all* bits be erased when tampering attempts are sensed. For instance, all-or-nothing [36] transforms could be used to ensure that even if very few of the encrypted bits are erased, there will be no way for the attacker to expose the secrets.

The evolving paradigm of physical unclonable functions (PUF) [22] however provides a very satisfactory solution to protect secrets of a device while the device is at-rest. PUFs **eliminate** the need for any kind of monitoring of a device at rest. Silicon PUFs [37] exploit uncontrollable statistical delay variations of connections and transistors etched on substrates, in each manufactured chip. Even though a manufacturer may fabricate many chips with identical masks, each chip would have a unique set of delays in its components, which even the manufacturer will not be able to measure with a high enough precision. In other words, each chip (or some components taken together in a chip) implements an "un-characterizable" (and therefore *unclonable*), unique, physical one-way function.

The *unique* PUF in each chip could be used along with challenge-response protocols for identification of devices. For instance, a chip with PUF $\mathcal{H}()$ generates *responses* $R_i = \mathcal{H}(C_i)$ for different *challenges* C_i. Or each device has a *unique* set of challenge-response pairs (CRP) $\langle C_i, R_i \rangle$. Thus, the key used for encrypting the secrets at rest could be $R_X = \mathcal{H}(X)$, where X is a randomly chosen "challenge" which is stored in the *clear* in the NVM in the device. When the device is at rest, there is no way for the attacker to determine R_X from X (as the only way to get R_X is by challenging the PUF which can be done only when the device is on). Further, by definition, PUFs are unclonable (even the manufacturer cannot synthesize two identical PUFs and use one to determine responses of the other). When the device is powered on, the device could challenge the PUF with X and obtain the response R_X, and then proceed to decrypt the stored secrets.

In practice PUFs may not be able to provide *reliable* CRPs as $\mathcal{H}$ may drift with temperature and aging. Gassend et al [22] argue that it is possible to employ error correction codes [38] to compensate for the drift of PUFs. The associated reduction in entropy of the hash function (due to error correction) could be addressed by increasing the complexity of PUFs.

4.1.3 Protecting Secrets During State-Transition

During the *in-use* state, the secrets may be decrypted and stored in RAM for use, but could be well protected by active monitors. During the *rest* state the secrets could be protected using rest-encryption (with PUFs). Perhaps the most vulnerable period of a trusted computer is during the transition from "in use" to "rest" state. The transition might typically involve 1) encryption of all keys for storage with a key and 2) *clean* erasure of all secrets stored in volatile memory.

While the second step may *sound* redundant at first sight (as the secrets in volatile storage will be lost when power supply is removed in any case), it is in fact required to ensure that the secrets temporarily stored in non-volatile memory do not leave a "footprint." Such footprints left behind in magnetic and solid state memories [39] could be used to decipher the previous contents of the memory, especially if they had been stored in a memory location for long periods. Safe erasure [40] of contents in magnetic and solid state memory (or removing all traces of their footprints) may require many *repeated* overwriting operations. The ability of the attacker to scavenge bits is also significantly enhanced by cooling the device (say by immersing it in liquid nitrogen).

To render his attack more worthwhile, the strategy of the attacker may be to induce a glitch attack that causes the computer to hang. Even if some sensors (which work independent of the CPU) detect intrusion attempts, they may not be able to perform the repeated overwriting operation needed for safe erasure. Thus immediate cooling following a glitch attack may be very productive for the attacker. The attacker may be able to extract all secrets from RAM. Another alternative to the glitch attack might be to use FIBs to suddenly cut off power supply to the CPU.

4.2 DOWN Policy

A simple security policy - **d**ecrypt **o**nly **w**hen **n**ecessary (DOWN) [13] could substantially reduce the susceptibility of trusted computers during the transition period. If contents of the RAM cannot be well protected following abnormal state transitions that may be induced by the attacker, a solution is to make sure that the RAM has very minimal information at *any point in time*. The DOWN policy recognizes the fact that most cryptographic operations have some inherent *atomicity*. At any point in time only one, or may be even a small part of a secret may be necessary for cryptographic computations.

For instance, if the secret to be protected is a RSA private exponent r (say of size 1024 bits), and n represent the RSA modulus, decryption of a some cipher text C involves modular exponentiation of C with r. Or $P = C^r \bmod n$. However, to perform the exponentiation, *only one bit of r is needed at any point in time* (say exponentiation using the square and multiply algorithm).

We could thus keep r encrypted at all times, and decrypt each bit *as and when necessary*.

4.2.1 Trusted Computers With DOWN

For trusted computer using DOWN, a single secret M could be used for encrypting all secrets stored in NVM. The secret M may be stored in some special volatile register in the CPU (which could be hidden even from the OS kernel) [23]. This register could be protected from scavenging attacks by repeated ones complementing of the register (say every few milliseconds are so) to ensure that it does not leave a discernible footprint. In addition to M the CPU may also have access to a hardware block cipher E (which is in fact commonly used in smart cards) which could generate many secrets $S_i = E_M(i)$ which could be used for encrypting the secret K_i stored in NVM.

At any point in time, the RAM can have utmost one S_i (and K_i) which the attacker may be able to scavenge from RAM (as RAM cannot possibly employ techniques like one-complementing to avoid footprints). With the DOWN policy, there is no explicit transition required for going from in-use to rest-state. Only the secret M needs to be protected at rest. With PUFs, even that need eliminated the secret could be encrypted with R_X and stored in the clear, in NVM - or R_X itself could be used instead of M.

5 KPSs With DOWN

The use of DOWN policy is more natural for KPSs where only one secret is needed at any point in time. Further, a *strict* implementation of the DOWN policy should ensure that at *all* times in the *entire life-cycle* of a device, only a part of the secret (or one of the k secrets for KPSs) is decrypted and stored in volatile RAM. This may not be feasible for even RSA[19] for instance, when the private and public key pairs are *generated*. However a strict implementation of the DOWN policy for KPSs does not pose any problems.

With the DOWN policy, the explicit transition is *eliminated*. The secrets stored in non-volatile memory are *always* encrypted. Thus the two main implications of the DOWN policy are

1. Non-volatile memory (NVM) used for storing k encrypted secrets does not need *any* protection.
2. By tampering with a device an attacker can expose no more than *one* KPS secret (in which process the device is rendered unusable). *Thus a n-secure KPS with k keys is rendered nk-secure!*

5.1 DOWN Complexity of KPSs

Earlier, in Section 1, we argued that *insecure* storage complexity (needed for storing P_{ij}s for I-HARPS is not an issue. With the DOWN policy, as even the k secrets are always stored encrypted in NVM, the secrets can also be stored in flash memory. So the size of k too is *not* an issue! Thus the DOWN policy hepls improve the security of any KPS in two ways

1. by allowing for large k, and
2. by rendering KPS nk or $O(k^2)$-secure

However, the main disadvantage of increasing k is that, in general, the *computational complexity* increases with k. With the DOWN policy, whenever a secret is needed, it needs to be fetched from NVM, decrypted, used, and flushed out of memory. Further, every time a secret is decrypted, the processor may need to switch to a secure kernel mode [23].

Thus with DOWN, the primary complexity is the *number of DOWN operations* needed to evaluate shared secrets (or establish security associations like K_{AB} between A and B). Each DOWN operation consists of 1) fetching from NVM, 2) switching to secure kernel mode, decryption of secret, 3) using the secret in some cryptographic algorithm, and 4) flushing the RAM (where the secret was stored) clear.

While increasing k is typically accompanied by increase in computational complexity for *most* KPSs, *this is not true for RKPSs*. As a numerical illustration of the effect of DOWN policy on KPSs, we shall compare three KPSs - Blom's scheme [2] and HARPS [12] and I-HARPS.

For Blom's scheme, *all* k secrets are needed for cryptographic computation involving discovery of shared secrets. However, for HARPS with $P = 15000$ and $k = 1000$ and I-HARPS with $P = 15000, k = 1000$, only about $\xi k \approx 67$ secrets are needed on an average! In other words, the following 6 systems have the same "DOWN complexity."

1. Blom's scheme with $k = 67$
2. HARPS with $P = 15,000, k = 1000$
3. I-HARPS with $P = 15,000, k = 1000$
4. HARPS with $P = 60,000, k = 2000$
5. I-HARPS with $P = 60,000, k = 2000, L = 1024$
6. I-HARPS with $P = 60,000, k = 2000, L = 2048$

All of them need 67 DOWN operations. In fact the cryptographic operation in *each* DOWN operation is actually significantly less complex with I-HARPS - it just involves evaluation of two hash functions. For Blom's scheme, each DOWN operation involves an evaluation of x^y mod P, where $x \in \mathbb{Z}_P$ and $1 \leq y \leq n$.

Table 1 provides a comparison of the 6 schemes. Also shown is the effect of increasing insecure storage on the security of I-HARPS (last row). For calculating the storage we have assumed that each P_{ij} value is 128-bits or 16 bytes (all keys are 128-bits long).

With DOWN, Blom's scheme is rendered $nk = 66 \times 67 = 4422$-secure. For HARPS and I-HARPS, the table shows the number of nodes the attacker needs to tamper with (assuming the attacker can expose one secret from each node - thereby destroying the node in the process), for $k = 1000$ and $k = 2000$, to discover *shared secrets* between nodes with some probability p_e. For HARPS with $k = 2000$, the attacker needs to destroy over 160,000 chips to ensure that he has a *one in a million* chance of compromising shared secrets between any two participants.

[19]It is not clear now as to how the DOWN policy could be used for other asymmetric ciphers that require operations other than modular exponentiation.

Table 1: Comparison of 6 KPSs with the same "DOWN complexity." All size - Blom's scheme with $k = 67$, $(P = 15,000, k = 1000)$ HARPS, $(P = 60,000, k = 2000)$ HARPS, $(P = 15,000, k = 1000)$ I-HARPS, and $(P = 60,000, k = 2000)$ I-HARPS (with $L = 1204$ and $L = 2048$) have DOWN complexity of 67.

Approach	nk - Number of nodes to be tampered with			WoD*	Storage
	$p_e = 1 \times 10^{-12}$	$p_e = 1 \times 10^{-6}$	$p_e = 0.5$		
Blom's Scheme	4422	4422	4422	66	0
HARPS $(P = 15,000, k = 1000)$	21,000	40,000	205,000	21	0
HARPS $(P = 60,000, k = 2000)$	84,000	160,000	815,000	84	0
I-HARPS $(P = 15,000, k = 1000)$	6,750,000	12,000,000	17,500,000	6750	16 MB
I-HARPS $(P = 60,000, k = 2000)$	27,000,00	48,000,000	70,000,000	13500	32 MB
I-HARPS $(P = 60,000, k = 2000)$	54,000,00	96,000,000	140,000,000	27000	64 MB
WoD* - Without DOWN Assurance, $p_e < 10^{-12}$					
Storage - Calculated as $k \times L \times 16$ bytes (128-bit keys and P_{ij}s.)					

With I-HARPS with $k = 2000$ and $L = 1024$ (or 32 MB insecure storage), the attacker has to destroy about *48 million* chips to achieve the same goal!

Thus with RKPSs it is possible to take advantage[20] of the practical, large NVM storage *without increasing the computational complexity* (as comparison between the two cases with $k = 1000$ and $k = 2000$ for HARPS and I-HARPS readily demonstrates).

With KPSs, secrets would be *renewed* periodically. In order to take part in renewal, a node with authenticate itself to the TA with *all* its k keys. So in order for the attacker to take part in key updates, the *minimum* requirement for an attacker is to discover *all* k secrets in a node[21].

Taking the effect of DOWN policy into consideration,

1. For HARPS with $k = 2000$, an attacker needs to destroy over 20 million nodes to have *one in a trillion* chance of discovering *all* 2000 secrets of some node
2. For I-HARPS with $k = 2000, L = 2048$, to achieve the same goal, an attacker has to destroy over *520 million* nodes (260,000 nodes *without* DOWN assurance).

While most recent works in the literature on random key predistribution schemes [5] - [10] (apart from [12] and [11]) are targeted towards their use in severely resource constrained sensor networks, with the DOWN policy, random KPSs, especially I-HARPS, become serious contenders *even for scenarios where resource constraint is not an issue*.

5.2 Practical Limits on k

While for random KPSs like I-HARPS, HARPS and RPS we can afford to increase k without increasing the DOWN complexity, increasing k also increases the complexity of the public function $F()$ used to determine the indexes of the shared keys. The complexity of the public function could be $O(k \log k)$ or $O(P)$ depending on the implementation of $F()$.

However, the public function $F()$ is just a random sequence generator. It does not need to satisfy strong cryptographic properties. The only requirement is that the output of $F()$ is reasonably close to a uniform distribution[22]. Even very low complexity block ciphers, with reduced Fiestel rounds (for example TEA [27] with 4-5 rounds instead of 32) could be used for this purpose.

Implementations of $O(k \log k)$ complexity would involve generation of k-length uniformly distributed random variables to generate the partial permutation $\{A_1 \cdots A_k\}$. However, comparison of two sequences $\{A_1 \cdots A_k\}$ and $\{B_1 \cdots B_k\}$ to determine intersections would have a complexity of $O(k \log k)$.

Implementations of $O(P)$ complexity could actually be more straight-forward. For instance if $L = 1024$ and $\xi = 1/16$, we could simply generate P uniformly distributed random integers $F(A) = R_1 \cdots R_P$ seeded by the node ID. The last four bits of R_i indicate if key i is assigned to node A (for instance only if the last 4 bits are all zeros - which will occur with a probability 1/16 - the desired probability for allocation of keys to each node). Similarly, 10 bits to the left of the 4 LSBs could indicate the ID of the node in system i (between 1 and 1024 - which will of course be ignored if the key i is not assigned to node A). Thus for HARPS or I-HARPS with $k = 1024, P = 16k = 16384$, we just need to generate 7168 32-bit random integers ($16 \times 14 = 224$ bits from seven such integers can be used for 16 14-bit values R_i).

Note that the later approach does not guarantee that each node has exactly k keys - some nodes may have more and some less. The *average* number of keys in each node would be k. However, for our analysis we have actually assumed this model (we assume that each key is allocated to a node with probability ξ).

5.3 HARPS vs I-HARPS

The significant improvement offered by I-HARPS over HARPS comes at a price. First is the need for extra storage, which may not be a crucial issue. In fact I-HARPS is rendered less computationally expensive than HARPS as for each shared key HARPS needs an average of $L/4$ additional hashes to be performed.

However I-HARPS loses much of the *versatility* of HARPS.

[20]Only random KPSs employing the "dimension" of uniqueness of intersection of large subsets. The LM-KPS cannot take advantage of the DOWN policy.

[21]However, this may not be sufficient [24] for the attacker

[22]If the distribution of the output of $F()$ is not uniform, it may result in marginal reduction of the efficiency of random KPSs.

With HARPS, the same set of secrets which are used for pairwise authentication, can also be used for broadcast authentication [41], instantaneous discovery of shared group secrets (or conference secrets) and non-instantaneous discovery of group secrets (or broadcast encryption [42]). All these are not possible with I-HARPS. Further, HARPS caters for trivial extensions to a tree-hierarchical deployment [43], while protecting higher levels of hierarchy from compromise of secrets in the lower levels of hierarchy.

6 Conclusions

We have presented a novel random key pre-distribution scheme which takes advantage of cheap and practically unrestricted insecure storage like flash memory cards that wireless hand held devices are expected to have to yield dramatic gains in security and over other key pre-distribution without increasing *secure* storage complexity and computational complexity. Specifically, the proposed schemes was shown to be an order of magnitude more efficient than Blom's scheme (which however cannot be implemented using pure symmetric cryptographic primitives), and over 300 times more efficient than other random key predistribution schemes of comparable complexity (which however do not need the additional insecure storage requirement).

We have also investigated the effect of a simple security policy, DOWN, on the security of the proposed scheme, and KPSs in general. In addition to being very efficient, the proposed scheme also lends itself well to efficient DOWN implementations.

Note that even with *no assurance of read proofing or tamper resistance*, I-HARPS with $k = 2000$ and a storage complexity of 64 MB can reasonably resist coalitions of up to 30,000 nodes (for resistance to attacks aimed at discovering shared secrets) and more than 260,000 nodes for synthesis attacks. Thus with very little support from technology for tamper resistance / read-proofing I-HARPS could provide a a secure alternative to PKI for ad hoc mutual authentication. One of our current research effort involves extension of I-HARPS to hierarchical deployments.

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] 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.

[3] T. Leighton, S. Micali, "Secret-key Agreement without Public-Key Cryptography,"*Advances in Cryptology* - CRYPTO 1993, pp 456-479, 1994.

[4] M. Dyer, T. Fenner, A. Frieze and A. Thomason, "On Key Storage in Secure Networks," *Journal of Cryptology*, **8**, 189–200, 1995.

[5] 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.

[6] H. Chan, A. Perrig, D. Song, "Random Key Predistribution Schemes for Sensor Networks," IEEE Symposium on Security and Privacy, Berkeley, California, May 2003.

[7] 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.

[8] S. Zhu, S. Xu, S. Setia S. Jajodia, "Establishing Pair-wise Keys For Secure Communication in Ad Hoc Networks: A Probabilistic Approach," Proc. of the 11th IEEE International Conference on Network Protocols (ICNP'03), Atlanta, Georgia, November 4-7, 2003.

[9] 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.

[10] 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.

[11] M. Ramkumar, N. Memon, R. Simha, "Pre-Loaded Key Based Multicast and Broadcast Authentication in Mobile Ad-Hoc Networks," Globecom-2003.

[12] M. Ramkumar, N. Memon, "An Efficient Random Key Pre-distribution Scheme for MANET Security," IEEE Journal on Selected Areas of Communication, March 2005.

[13] M. Ramkumar, "DOWN with Trusted Devices," submitted to New Security Paradigms Workshop (NSPW) 2005.

[14] D. Clarke, J-E Elien, M. Fredette, A. Marcos, R.L.Rivest, "Certificate chain discovery in SPKI/SDSI," Journal of Computer Security, vol. 9, no. 4, pp. 285 – 322(38), 2001.

[15] D. Boneh, M. Frankling, "Identity Based Encryption from the Weil Pairing," *Advances in Cryptology*, Crypto 2001,, Berlin, 2001, LNCS vol **2139**. *Advances in Cryptology: Proc. of Eurocrypt 84*, Lecture Notes in Computer Science, **209**, Springer-Verlag, Berlin, pp. 335-338, 1984.

[16] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, M. Yung, "Perfectly-Secure Key Distribution for Dynamic Conferences," Lecture Notes in Computer Science, vol 740, pp 471–486, 1993.

[17] T. Matsumoto, M.E.Hellman, “New Directions in Cryptography,” *IEEE Transactions on Information Theory*, **IT-22**(6), Dec. 1976, pp.644-654.

[18] D. R. Stinson, T. van Trung, “Some New Results on Key Distribution Patterns and Broadcast Encryption,” *Designs, Codes and Cryptography*, **14** (3) pp 261–279, 1998.

[19] L. Gong, D.J. Wheeler, “A Matrix Key Distribution Scheme,” *Journal of Cryptology*, **2**(2), pp 51-59, 1990.

[20] C.J. Mitchell, F.C. Piper, “Key Storage in Secure Networks,” *Discrete Applied Mathematics*, **21** pp 215–228, 1995.

[21] P. Erdos, P. Frankl, Z. Furedi, “Families of Finite Sets in which no Set is Covered by the Union of r Others,” *Isreal Journal of Mathematics*, **51**, pp 79–89, 1985.

[22] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, “Silicon Physical Random Functions,” Proceedings of the 9th ACM conference on Computer and communications security, pp 148-160, 2002.

[23] J.P. McGregor, R. B. Lee, “Protecting Cryptographic Keys and Computations via Virtual Secure Coprocessing,” ACM SIGARCH Computer Architecture News archive, **33**(1), March 2005.

[24] M. Ramkumar , “Safe Renewal of a Random Key Predistribution Scheme for Trusted Devices,” to be presented at the 6th IEEE Information Assurance Workshop (The West Point Workshop), United States Military Academy, West Point, New York, June 2005.

[25] A. Pfitzmann, B. Pfitzmann, M. Schunter, and M. Waidner, “Mobile User Devices and Security Modules: Design for Trustworthiness”; IBM Research Report RZ 2784 (#89262) 02/05/96, IBM Research Division, Zurich, Feb. 1996.

[26] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, T. Rabin, “Tamper Proof Security: Theoretical Foundations for Security Against Hardware Tampering,” Theory of Cryptography Conference, Cambridge, MA, February 2004.

[27] R. Needham, D. Wheeler, “TEA - A Tiny Encryption Algorithm,” Fast Software Encryption, 1994.

[28] R. Anderson, M. Kuhn, “Low Cost Attacks on Tamper Resistant Devices,” IWSP: International Workshop on Security Protocols, Paris, April 1997.

[29] S. Skorobogatov, R. Anderson, “Optical Fault Induction Attacks,” Cryptographic Hardware and Embedded Systems Workshop (CHES-2002), LNCS 2523, Springer-Verlag, ISBN 3-540-00409-2, pp 2–12.

[30] P. Kocher, “Differential Power Analysis,” Advances in Cryptology, CRYPTO 1999, Springer LNCS series, Vol 1666, pp 388–397.

[31] M.G Karpovsky, K. Kulikowski, A. Taubin, ” Robust Protection Against Fault-Injection Attacks of Smart Cards Implementing the Advanced Encryption Standard”. T Proc. Int. Conference on Dependable Systems and Networks (DNS 2004), July, 2004.

[32] O. Kommerling, M. Kuhn, “Design principles for tamper-resistant smart-card processors,” Proceedings of the Usenix Workshop on Smartcard Technology, pp. 9–20, 1999.

[33] P. Laackmann, B. Meier, H. Nguyen, Infineon Technologies, “Revolution In The Smart Card Security,” available at http://www.epn-online.com/page/11955/revolution-in-the-smart-card-security-physical-attacks.html.

[34] Y. Ishai, A. Sahai, D. Wagner, “Private Circuits: Securing Hardware against Probing Attacks,” CRYPTO 2003, Santa Barbara, CA, Aug 2003.

[35] IEEE P1619 - Standard Architecture for Encrypted Shared Storage Media, available at http://siswg.org/docs/.

[36] R. Rivest, “All-or-nothing encryption and and the package transform,” Fast Software Encryption 1997, LNCS **1267**, E. Biham Ed., Springer-Verlag, 1997.

[37] D. Lim, “Extracting Secret Keys from Integrated Circuits,” Masters Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 2004.

[38] F. M. Williams, N. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Press, 1977.

[39] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley and Sons, 2001.

[40] P. Gutman, “Secure Deletion of Data from Magnetic and Solid-State Memory,” Sixth USENIX Security Symposium, San Jose, California, July 1996.

[41] M. Ramkumar, “Broadcast Authentication With Hashed Random Preloaded Subsets,” ePrint Archive, 2005/134, May 2005.

[42] M. Ramkumar, N. Memon, “A DRM Based on Renewable Broadcast Encryption,” to be presented at the Visual Communications and Image Processing (VCIP) Workshop, Beijing, China, July 2005.

[43] M. Ramkumar, N. Memon, “A Hierarchical Key Predistribution Scheme,” Electro/Information Technology Conference, Lincoln, NE, May 2005.