Distributed Symmetric Key Management for Mobile Ad hoc Networks

Aldar C-F. Chan

Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
10 King's College Road, Toronto
Ontario M5S 3G4, Canada
aldar@comm.utoronto.ca

***Abstract*— Key management is an essential cryptographic primitive upon which other security primitives are built. However, none of the existing key management schemes are suitable for ad hoc networks. They are either too inefficient, not functional on an arbitrary or unknown network topology, or not tolerant to a changing network topology or link failures. Recent research on distributed sensor networks suggests that key pre-distribution schemes (KPS) are the only practical option for scenarios where the network topology is not known prior to deployment. However, all of the existing KPS schemes rely on trusted third parties (TTP) rendering them inapplicable in many ad hoc networking scenarios and thus restricting them from wide-spread use in ad hoc networks. To eliminate this reliance on TTP, we introduce distributed key pre-distribution scheme (DKPS) and construct the first DKPS prototype to realize fully distributed and self-organized key pre-distribution without relying on any infrastructure support. DKPS overcomes the main limitations of the previous schemes, namely the needs of TTP and an established routing infrastructure. It minimizes the requirements posed on the underlying networks and can be easily applied to the ad hoc networking scenarios where key pre-distribution schemes were previously inapplicable. Finally, DKPS is robust to changing topology and broken links and can work before any routing infrastructure has been established, thus facilitating the wide-spread deployment of secure ad hoc networks.**

Keywords — distributed cryptographic protocol, key pre-distribution, ad hoc network, cover-free family, probabilistic method, privacy homomorphism.

I. Introduction

As mobile ad hoc networks edge closer toward wide-spread deployment, security issues have became a central concern and are increasingly important. In fact, ad hoc networks cannot be used in practice if they are not secure, for example, in applications like emergency rescue and battlefield communication; if no security mechanism is used, an adversary can easily thwart the network establishment. Due to their inefficiency, asymmetric/public key cryptosystems, for example RSA, are unsuitable for ad hoc networks where there are constraints on computation and energy [10]. In fact, symmetric key systems, like DES, AES and keyed hash functions, are still the major tools for communication privacy and data authenticity in most networks. To provide secure communication for any group of nodes using symmetric key cryptography, these nodes need to share a common secret key[1].

Most security requirements, such as privacy, authenticity and integrity, can be addressed by building upon a solid key management[2] framework. In fact, a secure key management scheme is the prerequisite for the security of these primitives, and thus essential to achieving secure infrastructure in ad hoc networks. However, none of the existing key management schemes seem to be satisfactory for an ad hoc network due to the unique properties of the latter. The major limitation of these schemes is that most of them rely on a trusted third party (TTP), thus not fulfilling the self-organization requirement of an ad hoc network. In this paper, we present protocols for a number of parties to jointly generate key distribution patterns, a process which is usually conducted by TTP. At the end of the computation, each party holds a set of secrets/keys (with which it can compute a single group key with any group of other parties in a non-interactive manner) and none of the other parties know all the keys it holds.

By definition [30], an ad hoc network is peer-to-peer and does not rely on any fixed infrastructure; instead, it is self-creating on the spot and all networking functions (such as routing, mobility management, etc..) are performed by the nodes themselves in a self-organized manner. The challenge of designing key management protocols for ad hoc networks thus lies in establishing a secure communication infrastructure, before any routing fabric[3] has been established and in the absence of any trusted authority or fixed server, from a collection of mobile nodes which share no pre-initialized secret information and have no prior contact with each other.

In general, there are three classes of key management, namely key distribution, key agreement and key pre-

[1]This common key, a.k.a. private key, is used as one of the inputs to the encryption (and the corresponding decryption) algorithm. In symmetric-key system, the same key is used for both encryption and decryption.

[2]In this paper, when we mention "key management", we are referring to "symmetric key management".

[3]It is reasonable and necessary to have security infrastructure established well before any route is set up, otherwise an adversary can inject or modify any route information during the route establishment process to mislead other nodes to preclude any routing fabric setup.

distribution. When used in an ad hoc network, each one has its limitations.

The traditional Internet style key distribution protocols, for example Kerberos [21], relying on online trusted third parties (TTP) to distribute session keys to nodes are infeasible for ad hoc networks because the TTP may be out of reach or not available to some of the nodes or during certain times for a number of reasons. These include communication range limitations, node movements, network dynamics and unknown network topology prior to deployment. There exist a number of proposals for ad hoc networks [12], [17], [33] which attempt to increase the availability of the key distribution service by replicating the online key server to a subset of nodes arranged arbitrarily or hierarchically. However, the performances of these schemes, in terms of efficiency and scalability, are still not satisfactory. Besides, these schemes still need TTP; compromising the TTP compromises all the keys it issues.

On the other hand, contributory key agreement protocols [2], [3], [8], [19], [20], [24], in which each node contributes an input to establish a common secret (which is a function of all nodes' inputs) through successive pairwise message exchanges among the nodes in a secure manner using the 2-party Diffie-Hellman exchange [13], are not practical to ad hoc networks either. These protocols are fully distributed and self-organized without needing any TTP, but they are not robust to changing topology or intermittent links commonly occurring in an ad hoc network. In order to successfully establish a key, these protocols strictly require the underlying networks to either support broadcasting or have a relatively time-invariant topology of certain forms. Usually, all the nodes need to be online before the key establishment process is completed; if any node leaves in the midst due to link or battery outage, no common key would be established and the remaining nodes need to re-run the process from scratch. Obviously, these requirements could never be satisfied once an ad hoc network has been deployed. Besides, key agreement is not computationally efficient. The worst is: to support these pairwise exchanges or broadcasts in an ad hoc network, a routing infrastructure usually needs to be established because it is possible that two nodes may not reach each other and need the intermediate nodes to relay messages. As mentioned earlier, we need to establish a secure infrastructure before any route is set up to avoid disturbances from an adversary. Hence, the group key agreement approach without further improvement is not practical for ad hoc networks. On the other hand, the pairwise key agreement approach is not scalable due to the need of frequent interactive rekeying despite key freshness.

A number of recent works demonstrate that the key pre-distribution scheme (KPS), independently introduced by Blom [4] and Matsumoto et. al. [23], offers practical and efficient solutions to the key management problem in a variety of models including conferencing [6], [25], broadcast/multicast [9], [16], [18], [22], [26], and sensor networks [11], [15]. In KPS, an offline TTP pre-initializes each node in a set $\mathcal{S}$ with some secret information (a set of long-lived keys) with which any subset of nodes ($\mathcal{S}_i \subseteq \mathcal{S}$) later on can find or compute non-interactively a common session key which is secret from a collusion of at most a certain number of nodes outside $\mathcal{S}_i$. Two works on KPS for distributed sensor networks [11], [15] further suggest that KPS would be the only practical option to date for key management in networks whose topology is unknown prior to deployment or changes fast after deployment. However, their schemes, and in fact all the existing KPS, still rely on TTP in the initialization phase to pre-load each node with the long-lived keys and the corresponding key identifiers. Clearly, compromising the TTP compromises the security of the whole network. This limitation renders them inapplicable in many ad hoc network scenarios although the TTP is only needed during the initialization and can be offline for the rest of the time.

In an ad hoc network, the nodes or users (sharing no prior secret information) may just meet and get to know each other on the spot where there is likely no single proxy or TTP they all can trust for key pre-distribution. For example, some cases of emergency rescue, battlefield communication and conferencing. Since the nodes (at least in these cases) may have no knowledge about whom they will meet prior to network deployment, it is not possible for them to contact the same TTP in advance for offline initialization. Randomly selecting a node or a group of nodes online to perform the tasks of TTP is not feasible either because of increased security risk — If the chosen node(s) is (are) compromised, all the subsequent communications will be left unprotected. As a result, sharing the key pre-distribution task (usually done by TTP) by all the nodes/users involved, that is, adopting the notion of distributed trust, seems to be the only plausible option. Each one is assumed not to fully trust each other for key pre-distribution; instead, each one contributes equally to the key pre-initialization process. We will focus on this line of solutions for the key management problem in ad hoc networks.

In this paper, we introduce a new paradigm called Distributed Key Pre-distribution Scheme (DKPS) which is a collection of distributed cryptographic protocols for a number of nodes sharing no prior secret to jointly realize the key pre-distribution function without relying on any TTP (but with results identical to a TTP-based KPS) . The basic idea of DKPS is that each node individually picks a set of keys from a large publicly-known key space following some procedure so that at the end, the key patterns of all the nodes satisfy the following exclusion property with a high probability: any subset of nodes can find from their key patterns at least one common key not covered by a collusion of at most a certain number of other nodes outside the subset. After checking with other nodes in a secure manner, if a node finds that its key pattern cannot satisfy the exclusion property, it will re-select the keys. Since the key selection process is memoryless, the node will most probably get a proper key pattern this time. In fact, with properly chosen parameters, most nodes can get a proper key pattern in the first round. Based on the novel combination of probabilistic method and privacy homomorphism, we construct a practical scheme which demonstrates the existence and feasibility of DKPS. To

the best of our knowledge, it is the first ever DKPS prototype. No similar scheme of this kind has appeared in the literature before.

There are three phases in our DKPS scheme, namely distributed key selection (DKS), secure shared-key discovery (SSD) and key exclusion property testing (KEPT). In the DKS phase, each node[4], using the procedure from a probabilistic cover-free family (CFF) construction, randomly picks keys from a large pre-defined and publicly-known set P to form its key-ring ($P_i \subset P$). We prove in this paper that with properly chosen parameters, there is a high probability guaranteeing that the exclusion property is satisfied. The SSD phase is the most subtle part of our scheme. In the key discovery phase, each node needs to find out with each one of the others which keys in their key rings are in common but not to leak any information about the keys that the other side does not have. We base the SSD protocol on homomorphic encryption and introduce a new scheme that fits our purpose here. To allow a node to test whether its set of keys satisfy the exclusion property, the information about the keys shared with other nodes is arranged in a data structure called incidence matrix through which we derive a simple testing algorithm KEPT.

The major contribution of this paper is that the notion of distributed key pre-distribution scheme (DKPS) is introduced and the first ever practical scheme to realize fully distributed and self-organized key pre-distribution is constructed, which overcomes the main limitations of the previous schemes — the needs of trusted third parties and established routing infrastructure. Our scheme neither relies on any infrastructure support nor requires any TTP. It is particularly suitable for use in ad hoc networks since it minimizes the requirements posed on the underlying networks and can be readily applied to the ad hoc network scenarios where key pre-distribution schemes were previously inapplicable. Furthermore, our scheme is robust to changing topology and broken links because the SSD phase can be interrupted or even stopped at any point of its operation and resume later on without any penalty on performance. Unlike most key agreement schemes, rekeying (which is a hassle) is not needed in our scheme when a member leaves or a set partition occurs. Finally, it can work well before any routing infrastructure has been established, thus facilitating the wide-spread deployment of secure ad hoc networks.

The rest of this paper is organized as follows. In the next section, a review of related works is given. In Section III, we discuss privacy homomorphism and give the construction of a new scheme (tailored for use in DKPS). Then, we present the details of the various stages of DKPS in Section IV. Finally, we discuss some application and performance issues in Section V and summarize the results in Section VI.

II. Related Works

In this section, we first review key distribution schemes, then key agreement schemes, and finally key pre-distribution schemes and key management in sensor networks.

[4]We will use "user" and "node" interchangeably in this paper.

Traditional key distribution protocols rely on infrastructure with online trusted third parties (TTP). An example of these is the well-known Kerberos [21] scheme which is widely used on the Internet. Kerberos is based on the seminal idea of Needham et. al. [27] in which each user is pre-initialized with a single key shared with an online server known as the key distribution center (KDC). When the users want to establish secure communication among themselves, each one of them has to obtain a new session key, encrypted with his pre-initialized key, from the KDC. There are also a number of schemes extending this approach to ad hoc networks [12], [17] by replicating the KDC in an arbitrary or hierarchical arrangement. The reliance on online TTP renders these schemes impractical for ad hoc network key management.

Diffie and Hellman [13] presented the first key agreement protocol for 2 parties, which is the well-known Diffie-Hellman key exchange. A number of natural extensions to their scheme have been proposed for n parties [2], [3], [8], [19], [20], [24]. All of these schemes are based on successively running the 2-party Diffie-Hellman key exchange on different message exchange topologies and sequences to achieve a common key. The earliest generalization attempt was by Ingemarson et. al. (ING) [19]. This protocol uses a ring topology in which pieces of key information are passed along the ring and the minimum number of rounds required is $n - 1$. Burmester et. al. introduced the first n-party key agreement protocol (BD) based on a tree [8]. They described protocols based on star, tree, broadcast and cyclic topologies. McGrew et. al. [24] used a one-way function tree (OFT), which is a binary tree, for the exchange of key information. Kim et. al. [20] investigated a number of different tree structures for group key agreement which are efficient with respect to a number of group operations such as member add, member delete, group merge and group partition. Becker et. al.[3] demonstrated 2 protocols, namely Hypercube and Octopus, which achieve the lower bounds on the round complexity and the number of exchanged messages respectively. These key agreement schemes are fully distributed and do not need any server or TTP. However, they are still not suitable for ad hoc networks for a number of reasons:

- An established routing fabric is needed to achieve broadcasts, which are used in some of the group key agreement protocols, conflicting with our goal to build a secure infrastructure to protect route establishment.
- Two nodes which need to exchange messages may not have direct contact with each other in an ad hoc network; the key agreement process cannot be completed in this case unless a routing fabric has already been in place.
- Due to node mobility and link outage, an ad hoc network has a relatively fast changing topology; in some cases, before any key can be established, the topology may have already changed completely and a node may lose its link to a previous neighbor. Whenever there is a membership change in the midst of a key agreement process, a re-run (from scratch) is needed for all the nodes. Hence, completing a successful key establishment may take long.

The first key pre-distribution scheme was given by Blom [4]. In this scheme each one of the n users receives a set of secret keys. From his/her secret keys, user i can compute a common key with any other user, say j. In other words any two users can compute a common key without interaction. The scheme is against a collusion of at most w other users. Blundo et. al. [6] generalized Blom's scheme to allow any t users (out of n users) to compute a common key non-interactively. They also showed that the total size of the secret keys stored at each user is optimal. Broadcast Encryption by Fiat and Naor [16] is another model of key pre-distribution. But this model assumes that a single sender broadcasts to multiple receivers and the sender knows all the keys of each receiver. Thus, it is not applicable for our purpose. This scheme is secure against a collusion of at most r receivers. Naor et. al. further developed this work using a tree structure for key allocation in [26] to make the scheme secure against collusion of any size. This is the Subset Difference method. Applying a hierarchical tree to Naor's work [26], Halvey and Shamir [18] reduced the key storage requirement of each receiver by almost a square root factor without significantly increasing other parameters (computation complexity and message overhead are of the same order of magnitude). Blundo et. al. [5] used information theory to derive the lower bound on the key storage requirement of each node in KPS under different models.

Carman et. al. analyzed a wide variety of approaches for key agreement and key distribution in sensor networks [10]. They analyzed the overhead of these protocols on a variety of hardware platforms. Perrig et. al. proposed SPINS [29], a security architecture for sensor networks. In SPINS, each sensor node shares with a base station a distinct pre-initialized secret key which is used for encrypting the session keys. This is identical to Needham's idea [27] except that the base station just needs to store a single root key k_{BS} and the secret key of each node is generated from the node's identity and k_{BS}. Eschenauer and Gligor [15] applied probabilistic key pre-distribution to distributed sensor networks. Chan et. al. [11] further improved the security of this work [15] by using multiple paths for Eschenauer's indirect key establishment. These two pieces of work also suggested that key pre-distribution is the only practical option for the key management problem in scenarios where network topology is not known prior to deployment, such as sensor networks and ad hoc networks.

Except key agreement protocols, all these existing schemes rely on a trusted third party to a more or less extent for distributing keys online or offline in a centralized manner.

III. Privacy Homomorphism

Privacy Homomorphism (PH), introduced by Rivest et. al. [31], is an essential element for constructing our secure shared-key discovery protocol (SSD) in Section IV-B. In this section, we will discuss what a private homomorphic encryption is without giving the formal mathematical definition, and then the requirements posed by our DKPS construction on the properties of PH. We will also discuss why the existing PH schemes are inapplicable for DKPS. Then we present a new PH scheme — MRS, which is based on a scheme in [31].

Basically, PH is a special class of encryption functions which allow the encrypted data to be operated on directly without needing any knowledge of the decryption functions. For example, suppose $E_K(\cdot)$ is a PH encryption function with key K and $D_K(\cdot)$ is the corresponding decryption function. Then for all K, given the ciphertexts[5] of two plaintexts x and y (i.e. $E_K(x)$ and $E_K(y)$), we can easily compute $E_K(x+y)$ (the encrypted ciphertext/value of the sum of x and y) from $E_K(x)$ and $E_K(y)$ using a certain algorithm $\mathcal{A}dd(\cdot,\cdot)$ without needing to decrypt them first. (i.e. We do not need to know what $D_K(\cdot)$ is.) This is an additive PH — the addition of plaintext data in their encrypted forms giving an encrypted sum. There is also multiplicative PH, that is, an algorithm $\mathcal{M}ulti(\cdot,\cdot)$ exists for $E_K(\cdot)$ to find $E(xy)$ from $E(x)$ and $E(y)$ without any knowledge of $D_K(\cdot)$. An example is the well-known RSA.

Regarding to the adversary attacks, our SSD protocol just needs an encryption scheme secure to ciphertext-only attacks, that is, it is not possible for an adversary, given only the ciphertexts, to find the corresponding plaintexts or the key. To ensure security and proper operation, a PH scheme $E_K(\cdot)$ (and the corresponding $D_K(\cdot)$) for use in our SSD protocol needs to further satisfy the following three properties:

1) **[Additive]**: Given $E_K(x)$ and $E_K(y)$, a computationally efficient algorithm $\mathcal{A}dd(\cdot,\cdot)$ exists such that $E_K(x+y) = \mathcal{A}dd(E_K(x), E_K(y))$, that is, $E_K(x+y)$ can be found easily without needing to know what x and y are.
2) **[Scalar Multiplicative]**: Given $E_K(x)$ and t, a computationally efficient algorithm $s\mathcal{M}ulti(\cdot,\cdot)$ exists such that $E_K(t \cdot x) = s\mathcal{M}ulti(E_K(x), t)$, that is $E_K(t \cdot x)$ can be found easily without needing to know what x is.
3) **[Non-trivial Zero Encryption]**: Suppose $x_1 + y_1 = 0$ and $x_2 + y_2 = 0$ (usually in modular arithmetic). If $x_1 \neq x_2$ and $y_1 \neq y_2$, then
   - $\mathcal{A}dd(E_K(x_1), E_K(y_1)) \neq \mathcal{A}dd(E_K(x_2), E_K(y_2))$,
   - $D_K(\mathcal{A}dd(E_K(x_1), E_K(y_1)))$ $= D_K(\mathcal{A}dd(E_K(x_2), E_K(y_2))) = 0$.

   That is, $E_K(0)$ should have a number of different possible representations in the ciphertext domain, depending on the processing on the encrypted data that results it. But all of these representations are decrypted back to 0 (The decryption is a many-to-one mapping.).

The first two properties are needed to achieve the goal of the SSD protocol — to allow two users to find out common keys in their key rings but not to leak out to the other side any information of the keys outside the common intersection of the two key rings. In fact, the second property is implicitly included in the first property (that is, once the first property is satisfied, the second property is also satisfied) because $s\mathcal{M}ulti(E_K(x), t)$ can be achieved by running $\mathcal{A}dd(\cdot,\cdot)$ recursively t times on

[5]Raw data or message is known as plaintext, and encrypted data is known as ciphertexts.

$E_K(x)$ as $E_K(t \cdot x) = E_K(\sum_{i=1}^{t} x)$. The last property is needed to ensure that neither one of the two parties can use the exchanged messages to mount an exhaustive search for the keys of the other side. This is to guard against any compromised node. We will discuss more details in Section IV-B.

None of the existing schemes can fulfill all these requirements. Two of the five schemes proposed in [31] were found insecure to ciphertext-only attacks by Brickell et. al. [7]. The one that our work (MRS) is based on is one of the exceptions but it is still vulnerable to a possible attack shortcut. The other exception is RSA; it is secure to both ciphertext-only and known-plaintext attacks, but it can only support multiplications, not additions which are needed for SSD (the first property). There also exist other schemes [1], [7], [28], but they are either impractical [1], too computationally inefficient [28], or over-restrictive on the number of additions that can be done [7]. Regardless of all these, the major problem is that none of the existing schemes satisfy the last property — They all encrypt 0 to only one ciphertext representation; some of them even have a trivial encrypted zero which is publicly known. All of the existing schemes are inapplicable for the purpose of SSD. Hence, based on the work of Rivest et. al. [31], we introduce a new PH scheme which has improved security to fit the SSD protocol.

A. Original Rivest's Scheme

The original scheme of Rivest [31] works as follows:

- Let p and q be large primes and $n = pq$. (p, q) is the secret key and n is publicly known.
- **[Encryption]** $E_{(p,q)}(\cdot) : \mathbb{Z}_n \rightarrow \mathbb{Z}_n \times \mathbb{Z}_n$.[6]

$$E_{(p,q)}(m) = (m \bmod p,\ m \bmod q)$$

- **[Decryption]** Reduce the two components mod p and mod q respectively, then apply the Chinese Remainder Theorem (CRT) [32].
- **[Homomorphism]** This is both additive and multiplicative homomorphic. Componentwise addition and multiplication (mod n) of the ciphertexts give the encrypted values of the addition and multiplication of the corresponding plaintexts in $\mathbb{Z}_n$. Suppose $E_{p,q}(m_1) = (x_1,\ y_1)$ and $E_{p,q}(m_2) = (x_2,\ y_2)$, then
  - Addition:
    $E_{p,q}(m_1 + m_2)$
    $= \mathcal{A}dd(E_{p,q}(m_1),\ E_{p,q}(m_2))$
    $= (x_1 + x_2 \bmod n,\ y_1 + y_2 \bmod n)$
    $= (\hat{x}, \hat{y})$

    For those who knows the key (p,q), they can compute
    $(\hat{x} \bmod p,\ \hat{y} \bmod q)$
    $= ((m_1 + m_2) \bmod p,\ (m_1 + m_2) \bmod q)$
    $= E_{p,q}(m_1 + m_2)$. [7]
  - Multiplication:
    $E_{p,q}(m_1 m_2)$
    $= \mathcal{M}ulti(E_{p,q}(m_1),\ E_{p,q}(m_2))$
    $= (x_1 x_2 \bmod n,\ y_1 y_2 \bmod n)$
    $= (\tilde{x}, \tilde{y})$

    For those who knows the key (p,q), they can compute
    $(\tilde{x} \bmod p,\ \tilde{y} \bmod q)$
    $= (m_1 m_2 \bmod p,\ m_1 m_2 \bmod q)$
    $= E_{p,q}(m_1 m_2)$.

The security of this scheme is based on the difficulty of factoring composite numbers of large primes (same as RSA). Since an adversary does not know the factorization of n, he cannot use CRT to decrypt the ciphertexts. However, since both addition and multiplication are supported, there is a possible shortcut to mount a ciphertext-only attack as follows: Suppose we have the ciphertext $E(m)$. Using $\mathcal{M}ulti$ we can easily find $E(m^2)$. After that, we keep running $\mathcal{A}dd$ recursively until the result is equal to $E(m^2)$. Then the plaintext m is just the resulting number of rounds of $\mathcal{A}dd$ needed to achieve $E(m^2)$. Luckily, testing for equality is not always possible in the Rivest's scheme.

Beside this attack, the encrypted zero in the Rivest's scheme has vulnerable representations of the form $(ap,\ bq)$ where $a \in [0, q-1]$ and $b \in [0, p-1]$. When used in SSD, this would help any dishonest node, after running the SSD with another node, to determine the factorization of n and hence all the keys of the other side. The details of this threat are as follows: Suppose Bob wants to find out the common keys with Alice. In SSD, Alice sends to Bob encrypted pieces of information (about her keys) through which Bob can combine with his keys to form a list of encrypted values. This list that usually only Alice can decrypt contains encrypted zeros at places where Bob and Alice have a common key. When the SSD protocol is still in process, Bob would have no knowledge about p or q. However, after completing the protocol, he knows which of his keys are in common with Alice's (the purpose of SSD). Hence, he knows which one in the encrypted list is an encrypted zero. By running the Euclidean algorithm [32] on two encrypted zeros having different representations, he can easily determine p and q. Bob may have stored all the encrypted pieces from Alice and use the recovered p and q to decrypt them; hence, Bob can find all the keys of Alice. To handle this attack, a simple solution is to split the plaintext into pieces and multiply each one of them with different secret constants.

In the Rivest's scheme, if m is small, say smaller than p or q, it would be un-encrypted (trivial encryption). Furthermore, 0 is most likely encrypted to $(0, 0)$ if it results from processing large numbers or a large number of numbers (which is quite usual in SSD), that is, property (3) would not be satisfied. To fix all these problems, we derive a new PH scheme called modified Rivest's scheme (MRS). In MRS, we split the plaintext m into l pieces before applying it to the encryption function to ensure that the vulnerable form of encrypted zeros

[6] $\mathbb{Z}_n$ is integer modular n.

[7] The result follows because $n = pq$, hence for any integer z, $(z \bmod n) \bmod p = z \bmod p$ and $(z \bmod n) \bmod q = z \bmod q$

would not provide any advantage to an adversary, and to eliminate the notion of $\mathcal{M}ulti(\cdot,\cdot)$ (which is not needed in SSD) and the associated attacks. We also multiply these l pieces with random numbers[8] to avoid trivial encryption.

B. Modified Rivest's Scheme (MRS)

Given two large primes p and q and $n = pq$, the MRS encryption is of the form $E_{p,q,r_i,s_i}(\cdot) : \mathbb{Z}_n \rightarrow (\mathbb{Z}_n \times \mathbb{Z}_n)^l$ (where $1 \leq i \leq l$) and works as follows:

Encryption: To encrypt a message $m \in \mathbb{Z}_n$, the steps are as follows:

- Break down m into a number of, say l, arbitrary pieces/numbers — $(m_1, m_2, \ldots, m_l)$ in such a way that $m = \sum_{i=1}^{l} m_i \bmod n$.
- Randomly choose $r_i < p$ and $s_i < q$, $\forall i \in [1, l]$ (which are kept secret).
  Note that r_i's need to be different to avoid the attack due to the vulnerable form of encrypted zeros, so do s_i's.
- Apply the encryption function.

$$\begin{aligned} E_{p,q,r_i,s_i}(m) &= ((m_1 r_1 \bmod p,\ m_1 s_1 \bmod q), \\ &\quad (m_2 r_2 \bmod p,\ m_2 s_2 \bmod q), \\ &\quad \vdots \\ &\quad (m_l r_l \bmod p,\ m_l s_l \bmod q)) \\ &= ((x_1, y_1), (x_2, y_2), \ldots, (x_l, y_l)) \end{aligned}$$

Decryption: Given $c = ((x_1, y_1), (x_2, y_2), \ldots, (x_l, y_l))$, the decryption steps are as follows:

- Multiply the components by the corresponding r_i^{-1} and s_i^{-1} respectively (in $\mathbb{Z}_p$ and $\mathbb{Z}_q$), i.e.

$$\begin{aligned} &((x_1 r_1^{-1} \bmod p,\ y_1 s_1^{-1} \bmod q), \\ &(x_2 r_2^{-1} \bmod p,\ y_2 s_2^{-1} \bmod q), \\ &\quad \vdots \\ &(x_l r_l^{-1} \bmod p,\ y_l s_l^{-1} \bmod q)) \end{aligned}$$

- Use Chinese Remainder Theorem to find $m_1, m_2, \ldots, m_l \pmod n$.
- Sum up m_i's to recover m.

Encrypted Data Processing: Suppose $t \in \mathbb{Z}_n$ is a constant. Let $E_K(a) = ((x_1, y_1), \ldots, (x_l, y_l))$ and $E_K(b) = ((u_1, v_1), \ldots, (u_l, v_l))$ where $K = (p, q, r_i, s_i)$, $i \in [1, l]$ is the secret key. We can view $E_K(\cdot)$ as a l-tuple of 2-component vectors, then addition and scalar multiplication in the ciphertext domain are just componentwise vector addition and scalar multiplication (both in $\mathbb{Z}_n$) of the l-tuples of vectors, that is,

- Addition:
  $E_K(a+b) = (((x_1 + u_1) \bmod n, (y_1 + v_1) \bmod n), \ldots$
  $\ldots, ((x_l + u_l) \bmod n, (y_l + v_l) \bmod n))$
- Multiplication:
  $E_K(t \cdot a) = ((tx_1, ty_1), \ldots, (tx_l, ty_l)) \pmod n$

[8]These random numbers can be viewed as part of the secret key.

1) An Example of MRS: Let $p = 7$, $q = 11$, then $n = 77$. For simplicity, let $l = 2$, that is, each plaintext message is split into 2 smaller pieces. Let $r_1 = s_1 = 5$, $r_2 = s_2 = 3$.

Suppose there are two plaintext numbers in $\mathbb{Z}_{77}$: $a = 10$ and $b = 7$. It can easily be seen that $7a + b \equiv 0 \pmod{77}$ and $14a + 2b \equiv 0 \pmod{77}$. The MRS scheme runs as follows.

Encrypting a:
Decompose a into $a_1 = 4$ and $a_2 = 6$.

$$\begin{aligned} E(a) &= ((4 \times 5 \bmod 7,\ 4 \times 5 \bmod 11), \\ &\quad (6 \times 3 \bmod 7,\ 6 \times 3 \bmod 11)) \\ &= ((6, 9), (4, 7)) \end{aligned}$$

Encrypting b:
Decompose b into $b_1 = 3$ and $b_2 = 4$.

$$\begin{aligned} E(b) &= ((3 \times 5 \bmod 7,\ 3 \times 5 \bmod 11), \\ &\quad (4 \times 3 \bmod 7,\ 4 \times 3 \bmod 11)) \\ &= ((1, 4), (5, 1)) \end{aligned}$$

Computing $E(7a + b)$ in $\mathbb{Z}_{77}$:

$$\begin{aligned} E(7a+b) &= ((7 \times 6 + 1,\ 7 \times 9 + 4), \\ &\quad (7 \times 4 + 5,\ 7 \times 7 + 1)) \\ &= ((43, 67), (33, 50)) \end{aligned}$$

Computing $E(14a + 2b)$ in $\mathbb{Z}_{77}$:

$$\begin{aligned} E(14a+2b) &= ((14 \times 6 + 2 \times 1,\ 14 \times 9 + 2 \times 4), \\ &\quad (14 \times 4 + 2 \times 5,\ 14 \times 7 + 2 \times 1)) \\ &= ((9, 57), (66, 23)) \end{aligned}$$

For decryption,
$r_1^{-1} = 5^{-1} \equiv 3 \pmod 7$,
$s_1^{-1} = 5^{-1} \equiv 9 \pmod{11}$,
$r_2^{-1} = 3^{-1} \equiv 5 \pmod 7$, and
$s_2^{-1} = 3^{-1} \equiv 4 \pmod{11}$.

For use in the CRT,
$11^{-1} \equiv 2 \pmod 7$ and $7^{-1} \equiv 8 \pmod{11}$.

Decrypting $E(7a + b)$:
Multiplying $r_i^{-1} \pmod p$ and $s_i^{-1} \pmod q$,
$((43, 67), (35, 50))$

$$\begin{aligned} &\rightarrow ((43 \times 3 \bmod 7, 67 \times 9 \bmod 11), \\ &\quad (33 \times 5 \bmod 7, 50 \times 4 \bmod 11)) \\ &= ((3, 9), (4, 2)) \end{aligned}$$

Using CRT, the two components of $7a + b$ are:
$(3 \times 11 \times 2 + 9 \times 7 \times 8) \bmod 77 = 31$, and
$(4 \times 11 \times 2 + 2 \times 7 \times 8) \bmod 77 = 46$
Hence, $7a + b = (31 + 46) \bmod 77 = 0$

Decrypting $E(14a + 2b)$:
Multiplying $r_i^{-1} \pmod p$ and $s_i^{-1} \pmod q$,
$((9, 57), (66, 23))$

$$\begin{aligned} &\rightarrow ((9 \times 3 \bmod 7, 57 \times 9 \bmod 11), \\ &\quad (66 \times 5 \bmod 7, 23 \times 4 \bmod 11)) \\ &= ((6, 7), (1, 4)) \end{aligned}$$

Using CRT, the two components of $14a + 2b$ are:
$(6 \times 11 \times 2 + 7 \times 7 \times 8) \bmod 77 = 62$, and
$(1 \times 11 \times 2 + 4 \times 7 \times 8) \bmod 77 = 15$
Hence, $14a + 2b = (62 + 15) \bmod 77 = 0$

As can be seen from this example, different processing on the ciphertexts results in $E_K(0)$ which could have different representations in (mod n), but all these representations are decrypted back to 0. This is a nice property for the SSD protocol as one cannot easily guess whether the result of a computation is the ciphertext of 0. As indicated by this example, we may just need to set $s_i = r_i = c_i,\ \forall i$ (where c_i's are *distinct* randomly picked numbers) to reduce the temporary key storage requirement by half.

As suggested by this example, l does not need to be very large; in fact, $l = 2$ would be very sufficient. This can be explained as follows: Excluding $(0, 0)$ which is encrypted to $((0, 0), (0, 0))$ by MRS, there are $n - 1$ possible pairs in $\mathbb{Z}_n$ whose sum is 0. Each component of this pair can have $q \times p$ representations in $\mathbb{Z}_n \times \mathbb{Z}_n$. As a result, there are $(n - 1)n^2$ possible representations for an encrypted zero if $l = 2$. This number is sufficiently large for most applications. To mount a successful encrypted-zero vulnerable-form attack in MRS, an adversary needs to determine how m is split into m_i's. For $l = 2$, an adversary node would only have a probability of $1/n$ to guess correctly how m is split; that is reasonably sufficient.

2) Security of MRS: Just like the original Rivest's scheme, MRS is secure as long as factoring integer composite of large primes is difficult. It can be proved that MRS is at least as secure as the original Rivest's scheme using simple reduction. Unlike the original scheme, there is no notion of $\mathcal{M}ulti(\cdot, \cdot)$ in MRS, hence MRS is completely secure to the attack shortcut mentioned earlier. Testing for encrypted zero in MRS also seems to be unlikely, rendering it a suitable choice for use in the SSD protocol. More importantly, it is highly unlikely that a dishonest node can use the vulnerable form of encrypted zeros to mount an attack (mentioned earlier), after completing the SSD protocol with another node, to determine that node's SSD encryption keys (p and q) or any element in that node's key set it is not supposed to know.

IV. Distributed Key Pre-distribution (DKPS)

In this section, we present the details of DKPS including its three main components, namely distributed key selection (DKS), secure shared-key discovery (SSD) and key exclusion property testing (KEPT). Recall that we let each node randomly pick keys into its key ring in the DKS phase, then it can use the SSD protocol to find out which keys it has are in common with another node, and finally it can test by looking at the incidence matrix whether its keys satisfy the exclusion property. Based on these keys found using DKPS, any group of nodes can compute non-interactively a group key which is secure from any collusion of bounded size.

A. Distributed Key Selection (DKS)

In the DKS phase, each node randomly picks keys from the universal set (which is publicly known) to form its key ring using a certain procedure to ensure that the exclusion property is satisfied. As the majority of the existing KPS schemes [4], [6], [9], [16], [22], [25] can be considered as a special case of the cover-free family (CFF) [14], we will formulate the exclusion property of DKPS using CFF. The generalized definition of CFF is as follows:

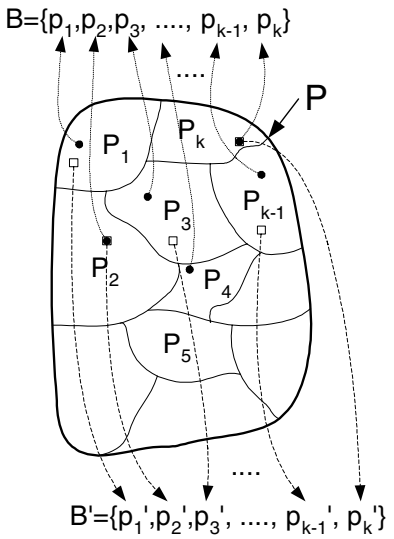

Fig. 1. DKS based on probabilistic CFF construction

Definition 1: Let P be an N-set of points $\{p_1, p_2, \ldots, p_N\}$ and $\mathcal{B}$ be a set of subsets (called blocks) of P (i.e. $X \subseteq P,\ \forall X \in \mathcal{B}$). Also let N and T denote $|P|$ and $|\mathcal{B}|$ respectively. Then the set system $(P, \mathcal{B})$ is called a $(w, r; d) - CFF(N, T)$ (cover-free family) if, for any w blocks $X_1, \ldots, X_w \in \mathcal{B}$ and any other r blocks $Y_1, \ldots, Y_r \in \mathcal{B}$, we have

$$|(\bigcap_{i=1}^{w} X_i) \backslash (\bigcup_{j=1}^{r} Y_j)| \geq d,$$

where d is a positive integer.

Although CFF is a widely adopted benchmark for formulating the security property of key pre-distribution schemes, the well known constructions (by coding theory or design theory) are centralized. Since we wish to eliminate the reliance on TTP, these techniques are not applicable; instead, we use a probabilistic method to construct CFF in a distributed manner.

By constructing $(w, r; 1) - CFF$ over a publicly known universal key set P, each node or user can pick a block $X_i \subset P$ as its key ring. Based on the property of $(w, r; 1) - CFF$, any subset of nodes with size up to w can find at least one common key from their key rings and any collusion of nodes outside this subset (with size smaller than r) have no knowledge about the common key. Ideally, given a group of nodes of size m, our goal is to construct a $(m, r; 1) - CFF$ in a distributed manner. However, to allow higher tolerance for key revocation due to some compromised nodes, we usually construct a $(m, r; d) - CFF$.

Depicted in Figure 1 shows how two nodes in our probabilistic construction can individually pick keys to form their

key rings while the exclusion property of $(w, r; d) - CFF$ could still be satisfied. Suppose we want to construct a $(w, r; d) - CFF$ assuming the capacity of each node's key ring is k_B. The detail of the DKS construction is as follows:

Step 1: Select $k \leq k_B$ s.t. d divides k. (k and d could be hard-coded.)

Step 2: Form the universal key set P with size $N = kur$ where $u = k/d$. (P and k are publicly known.[9])

Step 3: Divide P into k partitions $P_1, P_2, \ldots, P_k$ each of size ur.

Step 4: Each node individually pick keys for his key ring to form $B = \{p_1, p_2, \ldots, p_k\}$ with each p_i randomly selected from the partition $P_i, 1 \leq i \leq k$. (Each p_i will follow a uniform distribution over all elements in P_i.)

Finally, this method yields a $(w, r; d) - CFF(N, T)$ with the following guarantee:

Theorem 1: The probability that this system is not $(w, r; d) - CFF(N, T)$ is at most e^{-t} if the following condition on T (the number of users) is satisfied:

$$T \leq e^{\frac{2k\left(2-\frac{d}{k}-e^{-\frac{d}{k}}\left(\frac{d}{kr}\right)^{w-1}\right)^2 - t}{w+r-1}}$$

Proof: For a fixed block B_1, we select $w-1$ other blocks B_i, $2 \leq i \leq w$ and r other blocks C_j, $1 \leq j \leq r$. For any $p_i \in B_1$, we have

$$\begin{aligned} q &= Pr\left\{p_i \notin \bigcap_{i=2}^{w} B_i \backslash \bigcup_{j=1}^{r} C_j\right\} \\ &= 1 - \left(\tfrac{1}{ur}\right)^{w-1}\left(1 - \tfrac{1}{ur}\right)^{r} \\ &\geq 1 - \left(\tfrac{1}{ur}\right)^{w-1} e^{-\frac{1}{u}} \end{aligned}$$

Let $X_i = \delta\left[p_i \notin \bigcap_{i=2}^{w} B_i \backslash \bigcup_{j=1}^{r} C_j\right]$ where $\delta[\ ]$ is the delta function, then

$Pr\{X_i = 1\} = q$, and $Pr\{X_i = 0\} = 1 - q$.

Let $X = \sum_{i=1}^{k} X_i$, and apply Chernoff bound to it, we have

$$\begin{aligned} Pr\{X > a\} &= Pr\left\{\sum_{i=1}^{k} X_i > a\right\} \\ &< e^{-\frac{2}{k}(kq+a)^2} \end{aligned}$$

Using the bounds obtained above , we have

$$\begin{aligned} &Pr\left\{\left|\bigcap_{i=1}^{w} B_i \backslash \bigcup_{j=1}^{r} C_j\right| < d\right\} \\ &= Pr[X > k - d] \\ &< e^{-\frac{2}{k}(k(1+q)-d)^2} \\ &\leq e^{-2k\left(2-\frac{d}{k}-e^{-\frac{d}{k}}\left(\frac{d}{kr}\right)^{w-1}\right)^2} \end{aligned}$$

There are T possible choices for B_1 and $\binom{T-1}{w+r-1}\binom{w+r-1}{r}$ possible combinations for B_i and C_j (where $2 \leq i \leq w$, $1 \leq j \leq r$) leading to a total of $T\binom{T-1}{w+r-1}\binom{w+r-1}{r} < T^{w+r-1}$. The probability that the system is not $(w, r; d) - CFF$ is bounded above by:

$$T^{w+r-1} \times e^{-2k\left(2-\frac{d}{k}-e^{-\frac{d}{k}}\left(\frac{d}{kr}\right)^{w-1}\right)^2}$$

Substituting the statement of the theorem, we can achieve an upper bound of e^{-t}. ■

As can be seen, there is a high probability that the exclusion property of CFF can be satisfied if each node follows the procedure of the DKS scheme presented. If k is reasonably large and N is properly chosen for a given T, the probability that the resulting key patterns from the above method is not CFF could be negligibly small. Most of the nodes would successfully get a good key set in the first round; only a very small fraction need to run the DKS twice or more.

B. Secure Shared-key Discovery (SSD)

After running the DKS phase, each node has already share some common keys with another node, but it just does not know which keys in its key ring are in common with which node. This is to be done in the shared-key discovery phase.

In an usual KPS with TTP, the TTP gives each node a key identifier for each key in the latter's key ring. Since only the TTP knows the mapping between the keys and the identifiers, each node is restricted to know only the part of the mapping corresponding to its keys and none of the rest. So the shared-key discovery in this case is very simple and secure — each node just broadcasts its key identifiers to others; only those who have the same key know which key a particular identifier is referring to. For others, they gain no knowledge about the keys they do not have. This of course cannot be used for DKPS since each user knows the mapping between the keys and the identifiers in the DKS phase. The challenge here is how the nodes let each other know which keys they have are in common without revealing to the others the keys outside the commonly known fraction. We call it the secure shared key discovery (SSD) or the secure set intersection computation.

A trivial method for SSD would be as follows: each node broadcasts a list $\alpha, E_{K_i}(\alpha), \ldots, E_{K_k}(\alpha)$ where α is a challenge and $K_i, \ldots, K_k$ are its keys. The decryption of $E_{K_i}(\alpha)$ with a proper key by a received node would reveal the challenge α and hence discover a shared-key K_i with the broadcasting node. This method is simple but not secure because any one of the receivers can mount an exhaustive search for a particular key. Due to the properties of the distributed CFF construction in the DKS phase, an eavesdropper can run the DKS procedure as if he is one of the legitimate nodes to create a key ring which is likely to have a large overlap with the key ring of the broadcasting node. The eavesdropper in this case does not need to run an exhaustive key search over the entire key space; instead, he just needs to run a key search over the key ring he creates using the DKS procedure.

Before going into our SSD construction using MRS discussed in Section III, we would like to give the definition of SSD which is as follows:

[9]For simplicity, P could just be $\mathbb{Z}_N$.

Definition 2: The SSD or secure set intersection problem is as follows: Two parties, say Alice and Bob, have two sets $A = \{a_1, a_2, \ldots, a_l\}$ and $B = \{b_1, b_2, \ldots, b_m\}$ respectively. Suppose Alice wants to know which elements in her set are also in Bob's set, that is, to find $A \cap B$, but neither Alice nor Bob wants to let the other know the elements outside the intersection. They neither want any eavesdropper can learn about any element in A or B. How can they exchange messages to find $A \cap B$ securely?

Recall that MRS is an encryption scheme satisfying the following four properties:

- It is secure to ciphertext-only attacks.
- Given $E_K(x)$ and $E_K(y)$, it is possible to find $E_K(x+y)$ from $E_k(x)$ and $E_K(y)$without needing to know K, x or y.
- Given $E_K(x)$ and a constant t, it is possible to find $E_K(t \cdot x)$ without needing to know K or x.
- $E_K(0)$ has a large number of different representations which are non-trivial.

Denote the encryption function of MRS by $E_K(\cdot)$ and the corresponding decryption function by $D_K(\cdot)$, then the SSD protocol built upon MRS is as follows:

[SSD Protocol Using MRS (SSD-MRS)]

Suppose Alice has the key set $A = \{a_1, a_2, \ldots, a_l\}$ and Bob has the key set $B = \{b_1, b_2, \ldots, b_m\}$. There are two copies of the following protocol run in parallel — one from Alice to Bob and the other from Bob to Alice. Shown below is how Bob can check with Alice which keys he has are in common with Alice's. All arithmetics are done in $\mathbb{Z}_n$ (for some chosen $n = pq$ by Alice, where p, q are primes).

1) Alice forms a polynomial:

$$\begin{aligned} f_A(x) &= (x - a_1)(x - a_2)\ldots(x - a_l) \\ &= x^l + A_{l-1}x^{l-1} + \ldots + A_1 x + A_0 \end{aligned}$$

and sends Bob encrypted coefficients of $f_A(x)$ (encrypted with $E_{K_A}(\cdot)$ where K_A is Alice's secret key), that is, $E_{K_A}(A_0), E_{K_A}(A_1), \ldots, E_{K_A}(A_{l-1})$.
Note that if b_i is equal to one of the a_j's ($1 \le j \le l$), $f_A(b_i) = 0$.

2) Bob sorts b_i so that $b_1 < b_2 < \ldots < b_i < \ldots < b_m$. Bob chooses a random r. Due to the homomorphic properties of $E_{K_A}(\cdot)$, Bob can compute $z_i = E_{K_A}(r f_A(b_i))$, for $1 \le i \le m$. Since $E_{K_A}(0)$, the encrypted zero of MRS, could have several different representations, Bob cannot mount an exhaustive search as he cannot know what $E_{K_A}(0)$ should look like.

3) Alice, on receiving all these z_i's, applies $D_{K_A}(\cdot)$ and can get $r f_A(b_i)$ for $1 \le i \le m$. Since Alice has no knowledge about r, she does not know what b_i is. What she knows is: if $b_i \in A$, $r f_A(b_i) = 0$. For $r f_A(b_i) \neq 0$, since there are $u+1$ unknowns in u non-linear equations (for some $u \le m$), it would not be possible for Alice to find b_i in the information theoretic sense.

4) Alice returns an m-bit bitmap with 1 at bits where $r f_A(b_i) = 0$ and 0 elsewhere to Bob. A "1" at the i-th bit indicates to Bob that Alice also has b_i.

There is no incentive for Alice to forge a flaw bitmap to deceive Bob because at the end of SSD, all the keys held by Bob (even those in the common fraction) will not be revealed to Alice. If Alice forges a bit corresponding a key that she does not have, she would be penalized indirectly because Bob would assume that Alice has that key and may send her a message encrypted with a session key computed using that key as a part. (Note that all the common keys in $A \cap B$ are involved in computing the session key for Alice and Bob.) In this way, Alice would not be able to decrypt messages from Bob. In fact, the other protocol (from Alice to Bob) run in parallel provides an auxiliary means for Bob to verify any forged bitmap from Alice.

C. Key Exclusion Property Testing (KEPT)

For a particular node, the pattern about how its keys are shared with other nodes can be represented in a binary data structure called incidence matrix. Through its incidence matrix, each node can test whether all its keys satisfy the exclusion property of a $(w, r; d) - CFF$ when the information about the keys shared with other nodes is available. If a node has k keys in its key ring and there are m other nodes, then its incidence matrix is an $m \times k$ binary matrix $\mathbf{A} = [a_{ij}]$ with:

$$a_{ij} = \begin{cases} 1, & \text{if the } j\text{-th key is shared with the } i\text{-th node} \\ 0, & \text{otherwise} \end{cases}$$

The positions of "1" in the i-th row of an incidence matrix of a node u represent all the keys node u shares with the i-th node. The positions of "1" in the j-th column represent all the other nodes sharing the j-th key with node u. The incidence matrix $\mathbf{A}$ of a $(w, r; d) - CFF$ is one in which the vector formed by taking bitwise-AND of any w rows of $\mathbf{A}$ has at least d positions of '1' different from the vector by taking bitwise-OR of any other r rows in $\mathbf{A}$. A simple and straightforward algorithm to test the exclusion property is as follows:

```
For x = 1 to w,
  For each one of the (m choose x) x-row-subset,
    Compute the bitwise-AND for the x rows, say a.
    For each one of the (m-x choose r) r-row-subset from
    the remaining m − x rows,
      Compute the bitwise-OR for the r rows, say b.
      If the number of "1" in the bitwise-AND of
      a and b > k − d,
        Fail to satisfy (w, r; d) − CFF and break
      Else
        Continue
Success to satisfy (w, r; d) − CFF
```

D. Exchanging Entries of the Incidence Matrices

After two nodes complete running the SSD protocol, they would know which keys they have are in common. If key j

is in common, each one of them can pass to the other the j-th column of its incidence matrix. This is possible without needing to reveal the original key because the list of keys exchanged in the SSD protocol is sorted. In this way, a node may collect enough key sharing information of the other nodes not in contact so that the node may not need to run the shared-key discovery with these nodes later on. This may not have significant improvement at the initialization phase where all the nodes have no complete picture of the shared-key patterns. However, once a group of nodes have completed the initialization, each one of these nodes would already have run the SSD with each other; a newly joined node may just need to run the SSD with a small number of nodes (not all of the original nodes) to fill up its incidence matrix. In a $(w, r; d) - CFF$ system, this number might be slightly larger than r, thus significantly improving the latency and communication complexity of the key management for adding new nodes.

E. Computing Group Key

After completing the DKPS protocols, each node can find from its key ring a subset of keys in common with any group of other nodes, through which the nodes can compute a single group key, and no collusion ($\leq$ a certain size) of nodes outside the group knows all these common keys. The group key is thus safe and secret from any collusion with bounded size. An example of the group key computation for use in DKPS could be a hash of the concatenation of all the common keys, that is, the group key $K_{group} = h(k_1||\ldots||k_i||\ldots||k_x)$, where $h(\cdot)$ is a hash function and all these k_i's belong to the common intersection of the key rings of all the nodes in the group. Nobody outside the group would know all these k_i's, and K_{group} is thus secret and safe.

Like any of the existing key pre-distribution schemes, say the Blom's scheme [4], the group key computation in DKPS can be done at the time of use without needing any further interaction among the nodes once the key pre-distribution and shared key discovery have been completed in the initialization phase. This is a very nice property because any node can compute an encryption key on its own, use it to encrypt messages and send the encrypted messages to the targeted receivers using a one-way channel (say broadcast), and each receiver, by looking at the recipient list of the encrypted message, can compute the decryption key itself without needing any further interaction with the sender. This property is particularly important to the battlefield communication where some of the nodes need to be in silent mode (only receiving but not transmitting) to avoid being detected by the enemies.

V. Discussions

The motivation of DKPS is to provide key management to cover some of the previously intractable ad hoc networking scenarios in which a group of nodes get to know each other on the spot and do not trust any single node to conduct the key pre-initialization. DKPS facilitates the notion of distributed trust and provides completely self-organized key management solution to ad hoc networks. Ideally, DKPS should be completed when an ad hoc network is first formed. In fact, this can be achieved in most scenarios in which all the nodes usually get together for identifying each other and briefing at the time of network formation; that is also when the nodes establish initial trust. DKPS can be completed simultaneously during that time. In fact, this requirement is not strictly necessary for DKPS as new nodes can be added or the DKPS can be halted and resume later on after the network is deployed.

The major strength of DKPS is that no trusted third party is needed and it works without relying on any routing infrastructure. When a node first join an ad hoc network, it can just start to pick its keys individually. By running SSD with its neighbors, the node can immediately establish secure links with all these neighbors through the common keys discovered in SSD. Once the SSD phase has been fully completed, the node can move to anywhere to establish secure links with its neighbors non-interactively. If node i is compromised, its neighbors revoke all node i's keys and can still find some keys in their key rings that node i does not have, and use them to establish a new key also non-interactively. As long as the number of compromised nodes is smaller than a certain threshold r, the network remains in secure communication.

In case the SSD phase has not been completed, when node i moves into a new area and finds new neighbors, by running SSD, he can establish new secure links as before. If node i meets a previously acquainted node, node i does not need to run SSD again because it has already known which keys in its key ring to use for computing the encryption key non-interactively. Hence, a secure infrastructure can be built without relying on any established routing infrastructure. In addition, the performance of DKPS is not affected by topology or neighborhood changes. Although the DKPS protocol is supposed to be completed at the start of network deployment, the SSD protocol can be interrupted in the midst and resume after deployment.

Comparing with group key agreement schemes, DKPS has the following differences from it:

- DKPS takes into account all possible subsets of the current group of nodes whereas key agreement just consider the current group of nodes. Hence, when there is a member-leave or set partition (which is common in ad hoc networks), group key agreement needs a complete re-run whereas the initial secrets distributed by DKPS can be used to find a new key for the new partition/neighborhood.
- Key agreement needs to be completed at the first contact[10] whereas some of the nodes in DKPS may escape in the midst of the computation and do it later without affecting the remaining nodes' computations. The intermittent connectivity in ad hoc networks may cause the group key agreement process to take long to converge.

[10]Whenever a node disappear in the midst of exchange, all the remaining nodes need to re-run from scratch.

- Most key agreement schemes work on a certain fixed topology and sequence of message exchanges. This reduce their efficiency and increase the number of steps needed to converge in ad hoc networks which have arbitrary and usually time-varying topologies. In contrast, DKPS has no such restriction.
- Compared to the pairwise key agreement approach, the storage requirement of DKPS is an order of magnitude smaller. For pairwise key agreement, the number of keys each node needs is $n-1$ where n is the total number of nodes and there are $\binom{n}{2}$ keys in the network.

Although the SSD protocol uses MRS in this paper, the SSD protocol is quite flexible and other homomorphic encryption schemes can also be used as long as they satisfy the requirements posed by SSD. Since the cover-free family (CFF) construction achieves perfect secrecy, the key storage requirement at each node may be large. In fact, the probabilistic construction used in this paper is already close to achieving the lower bound of CFF. To optimize the storage, linking the keys together using a one-way function is a possible choice, but this trades off the security from perfect secrecy to complexity theoretic secrecy.

VI. Conclusions

Key establishment is the bottleneck to providing secure infrastructure for ad hoc networks. Key pre-distribution schemes are believed to be the best option for ad hoc networks, but all of the existing schemes rely on a trusted third party, thus limiting its use in ad hoc networks. In this paper, we demonstrate the feasibility of distributing the task of a trusted third party to all the nodes through the construction of the first ever DKPS prototype. As future works, we will research for more efficient constructions of DKS (in terms of key storage) and SSD (in terms of computational and message overheads).

Acknowledgment

The author would like to thank Croucher Foundation, Hong Kong for financially supporting this work.

References

[1] N. Ahituv, Y. Lapid, and S. Neumann. "Processing Encrypted Data", Communications of ACM, vol. 20, no. 9, p. 777-780, Sept 1987.
[2] G. Ateniese, M. Steiner, and G. Tsudik. a "New multiparty authentication services and key agreement protocols", IEEE Journal on Selected Areas in Communications, 18, no. 4, p. 628-640, April 2000.
[3] K. Becker, and U. Wille. "Communication complexity of group key distribution", in *proc.* ACM CCS'98, Nov. 1998, p.1-6.
[4] R. Blom. "An Optimal Class of Symmetric Key Generation System", in Advances in Cryptology - Eurocrypt'84, LNCS vol. 209, p. 335-338, 1985.
[5] C. Blundo, and A. Cresti. "Space requirements for broadcast encryption", in Advances in Cryptology - Eurocrypt'94, LNCS vol. 950, p. 471-486, 1994.
[6] C. Blundo, A. De Santi, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung. "Perfectly-secure Key Distribution for Dynamic Conferences", in Advances in Cryptogology - Crypto'92, LNCS vol. 740, p. 471-486, 1993.
[7] E. F. Brickell and Y. Yacobi. "On Privacy Homomorphisms", in Advances in Cryptology - Eurocrypt'87, LNCS vol. 304, p. 117-125, 1988.
[8] M. Burmester, and Y. Desmedt. "A secure and efficient conference key distribution system", in *proc.* Eurocrypt'1994, May 1994, p. 275-286.
[9] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. "Multicast security: A taxonomy and some efficient constructions", in Proc. IEEE INFOCOM99, 1999.
[10] D. W. Carman, P. S. Kruus, and B. J. Matt. "Constraints and Approaches for Distributed Sensor Security", NAI Labs Technical Report #00-010, September, 2000.
[11] H. Chan, A. Perrig, and D. Song. "Random Key Predistribution Schemes for Sensor Networks", to appear in proc. IEEE Symposium of Privacy and Security 03, May 2003.
[12] B. DeCleene, L. Dondeti, S. Griffin, T. Hardjono, D. Kiwior, J. Kurose, D. Towsley, S. Vasudevan, and C. Zhang. "Secure Group Communications for Wireless Networks", in proc. IEEE MILCOM01, Oct. 2001.
[13] W. Diffie, and M. E. Hellman. "New Directions in Cryptography", IEEE Trans. Inform. Theory, IT-22, p. 644-654, Nov. 1976.
[14] P. Erdös, P. Frankl, and Z. Füredi. "Families of finite sets in which no set is covered by the union of r others", Israel Journal of Mathematics vol. 51 (1985), no. 1-2, p.75-89, 1985.
[15] L. Eschenauer, and V. D. Gligor. "A Key-Management Scheme for Distributed Sensor Networks", in proc. ACM CCS'02, Nov. 2002.
[16] A. Fiat, and M. Naor. "Broadcast Encryption", in Advances in Cryptology - Crypto'93, LNCS vol. 773, p. 480-491, 1994.
[17] S. P. Griffin, B. T. DeCleene, L. R. Dondeti, R. M. Flynn, D. Kiwior, and A. Olbert. "Hierarchical Key Management for Mobile Multicast Members", 2002.
[18] D. Halevy, and A. Shamir. "The LSD Broadcast Encryption Scheme", in Advances in Cryptology - Crypto'2002, LNCS vol. 2442, p. 47-60, 2002.
[19] I. Ingemarsson, D. T. Tang, and C. K. Wong. "A conference key distribution system", IEEE Trans. on IT, vol. 28, no. 5, p. 714-720, Sept. 1982.
[20] Y. Kim, A. Perrig, and G. Tsudik. "Communication-efficient group key agreement", in *proc.* IFIP SEC'01, 2001.
[21] J. Kohl, and B. Neuman. "The Kerberos Network Authentication Service (V5)", RFC 1510, September, 1993.
[22] R. Kumar, S. Rajagopalan, and A. Sahai. "Coding constructions for blacklisting problems without computational assumptions", in Crypto'99, 1999.
[23] T. Matsumoto, and H. Imai. "On the key predistribution systems: A practical solution to the key distribution problem", in Advances in Cryptology - Crypto'87, LNCS vol. 293, p. 185-193, 1988.
[24] D. A. McGrew, and A. T. Sherman. "Key establishment in large dynamic groups using one-way function trees", May 1998.
[25] C. J. Mitchell, and F. C. Piper. "Key Storage in Secure Networks", Discrete Applied Mathematics, vol. 21, p. 215-228, 1988.
[26] D. Naor, M. Naor, and J. Lotspiech. "Revocation and Tracing Schemes for Stateless Receivers", in Advances in Cryptology - Crypto'01, LNCS vol. 2139, p. 41-62, 2001.
[27] R. Needham, and M. Schroeder. "Using Encryption for Authentication in Large Networks of Computers", in Commu. of ACM, vol. 21, no. 12, p. 993-999, 1978.
[28] P. Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes", in Advances in Cryptology - Eurocrypt'99, LNCS vol. 1592, p. 223-238, 1999.
[29] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. "SPINS: Security Protocols for Sensor Networks", in proc. ACM MOBICOM'2001, July 2001.
[30] C. E. Perkins. Ad Hoc Networking. Addison Wesley Professional, Dec. 2000.
[31] R. L. Rivest, L. Adleman, and M. L. Dertouzos. "On Data Banks and Privacy Homomorphisms", in Foundations of Secure Computation, Academic Press, p. 169-179, 1978.
[32] D. R. Stinson. Cryptography: Theory and Practice. CRC Press, 1995.
[33] M. Striki, and J. S. Baras. "Key Distribution Protocols for Multicast Group Communication in MANETs", University of Maryland ISR Technical Report #TR2003-17, 2003.