AN AUTHENTICATED KEY AGREEMENT SCHEME FOR SENSOR NETWORKS

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By

Mee Loong Yang

School of Engineering

October 2014

Copyright

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

_____

Signature of candidate

Acknowledgements

I would like to express my deep and sincere gratitude to my primary supervisor, Professor Adnan Al-Anbuky of the School of Engineering, AUT University, for his guidance throughout. My thanks and appreciation also to my second supervisor Dr William Liu of the School of Computer and Mathematical Science, AUT University for his great support and helpful criticisms.

I am grateful to Professor Ajit Narayanan, who as Head of School of Computer and Mathematical Sciences, encouraged and provided me with every resource and support to undertake this study. My thanks also to the Research committee of the School of Computer and Mathematical Sciences for support in funding for publications and conference presentations.

I am deeply grateful to my lovely my wife for her encouragement, support and understanding while I am away, leaving her to hold the fort. Last, but not least, I cannot thank my parents enough because it was their vision and sacrifice that allowed me an education in the first place.

Abstract

In wireless sensor networks, the messages between pairs of communicating nodes are open to eavesdropping, tampering, and forgeries. These messages can easily be protected using cryptographic means but the nodes need to share a common secret pairwise key. This thesis proposes a new scheme, the Blom-Yang key agreement (BYka) scheme, that enables pairs of sensor nodes in large networks to compute their pairwise keys quickly and efficiently. Prior to deployment, the Trusted Authority (TA), assigns each node their public *ID*s, and using its master keys, computes and stores in the nodes their private key-sets. When a pair of nodes need to obtain their pairwise keys, they exchange their public key identifier *ID*s which are just 16-bit integers. Using the counterpart's *ID* with its own set of private keys, the nodes are able to compute a large common pairwise key, but only if they have obtained their keying material from the same TA. Hence, the scheme is also mutually authenticating. The computations use simple arithmetic operations which are fast and efficient, easily undertaken by sensor devices which have limited computational, memory, and energy resources. For example, it is able to compute keys of 128 bits in 279 milliseconds in the MICAz mote, requiring 1170 bytes of memory to store the private keying material. Similar key agreement schemes, already widely used in computer networks, use public key cryptographic algorithms which require computationally expensive mathematical operations, taking much longer time, and requiring much more resources.

The security of the BYka scheme is based on the difficulty of obtaining information

about the private-public-master-key associations (PPMka). The private keys in each node are computed by the TA using all the permutations of its multiple master keys and the node's public keys operating over a small prime field, and then stored in a random order in the node. If these are captured, the private keys cannot be used directly as the adversary would first have to discover the PPMka. The analysis showed that, with suitable keying parameters, even if sufficient number of private keys are stolen, an adversary with powerful computing resources would need to expend an infeasibly large amount of time and resources to try all the possible PPMka to break the scheme. The adversary may try to discover the PPMka by using pairs of captured nodes to compute their pairwise keys, but this would require the capture of tens of thousands of nodes. Alternatively, even when using the most efficient method, the adversary needs to try a large number of possibilities equivalent to security strengths of 80 to 192 bits. Overall, the adversary has only a small probabilistic chance of breaking the scheme. These analytical results were verified using computer simulated attacks and are used to provide some guidelines and tables for the selection of the keying parameters to meet implementation and performance requirements including computation times, memory availability, network sizes, and pairwise key sizes.

The proposed key agreement scheme is in effect a non-interactive identity-based scheme which uses the node's identity (*ID*) as its public key. This allows a node to encrypt messages to a target node once its *ID* is known. It can be used by nodes in dynamic, mobile and ad hoc situations to opportunistically send authenticated messages to each other when they are in range. A single message authenticated protocol (SMAP) using the BYka scheme as the cryptographic primitive is proposed. The speed, efficiency, and resilience of the BYka scheme would make it useful as the cryptographic primitive in other applications such as email and voice communications.

Contents

List of Tables

List of Figures

Glossary and Notations

BYka scheme	The Blom-Yang key agreement scheme
Capture threshold	Number of compromised nodes required to break Blom's scheme
HMAC	Hashed message authentication code
IBC	Identity Based Cryptography
Keyspace	The maximum number of possible keys
Master key	A secret $(m \times m)$ symmetric matrix known only to the TA
PKC	Public Key Cryptography
Public key	A $(m \times 1)$ vector unique to a node and is publicly known
Public key-set	The set of public keys assigned to a node
Public key *ID*	An integer representing the node's public key-set
Private key	A $(1 \times m)$ row vector, unique and secret to the node
Private key-set **S**	A set of private keys, unique and secret to the node
Pairwise key, K_{AB}	The shared secret key between nodes A and B
Pairwise key-set, R	A set of integers for forming the pairwise key K_{AB}
Pairwise session key	A short term secret key shared between a pair of nodes
PPMka	Private-Public-Master-key association
TA	Trusted Authority
Traitor node	A node in which the PPMka is known

Notations

ID – identity of a node, an integer unique to the node

K – private key, a $(1 \times m)$ row vector

M – master key, a secret symmetric $(m \times m)$ matrix belonging to the TA

N – the number of master keys

Q_o – the size of the private key-set in bytes

R – the set of integers for forming the pairwise key K_{AB}

S – the private key-set, the set of $N\eta$ private keys

V – public key, an $(m \times 1)$ column vector

Φ – Number of possible master key solutions

m – the size of the master key matrix

n_c – the number of captured nodes required to find a traitor node

η – number of public keys assigned to each node

p – prime modulus for key operations

q – prime modulus for public key operations only

s – public key seed value

Chapter 1

Introduction

Wireless sensor devices have the potential to play an important part in all kinds of monitoring applications due to their small physical size, low cost, and wireless communications. However their widespread acceptance, especially in sensitive applications, will not be fully realised unless users are confident of its security. This is especially important for sensitive applications such as intruder detection, production plant process monitoring, military applications, and patient health monitoring. A basic requirement is that an adversary cannot read, modify, or forge the messages between the nodes. This requires pairs of nodes to share a common secret pairwise key for use with established cryptographic tools. For large ad hoc sensor networks, a key establishment scheme which enables pairs of nodes to compute their own pairwise keys would be most suitable.

Schemes based on asymmetric keys are often called public key cryptographic (PKC) schemes such as DH (Diffie & Hellman, 1976) and RSA (Rivest, Shamir, & Adleman, 1978). They are used in computer networks and have been studied for application in low resourced sensor devices (Ugus, Westhoff, Laue, Shoufan, & Huss, 2007),(Grosschadl, Szekely, & Tillich, 2007),(Lederer et al., 2009). In these schemes, the nodes generate their own keying material. These PKC schemes use expensive mathematical operations.

The computation times are long (M. Liu, Wei, & Liu, 2009), and require additional mechanisms for entity authentication. Identity-based schemes using bilinear pairing cryptographic methods, which also use PKC algorithms, have also been proposed for sensor networks (Szczechowiak & Collier, 2009) (L. B. Oliveira et al., 2011). There is no need for a separate mechanism for authenticating the public keys as the Trusted Authority (TA) provides all the keying material.

Symmetric key establishment schemes do not use expensive mathematical operations. Here, the TA is responsible for providing all the pairwise keys. However, instead of computing and distributing them, it delegates the pairwise key computations to the nodes by providing them with the key computation algorithm and a unique share of the keying material. Pairs of nodes are able to compute their pairwise key using their keying material. Various symmetric key schemes have been proposed such as those requiring the help of intermediary nodes (Chan & Perrig, 2005) (Eschenauer & Gligor, 2002), using a shared key with the base station (Zhu, Setia, & Jajodia, 2006), using a public database to obtain pairwise keys (Leighton & Micali, 1994), the polynomial based scheme in (Blundo et al., 1995), and the Blom's scheme (Blom, 1984). Of these, the most suitable for mobile ad hoc network are the schemes due to Blundo and Blom. The Blundo's two party scheme is equivalent to the Blom's scheme.

1.1 Motivation

The Blom's scheme uses simple mathematical operations allowing it to be used in low resourced devices, for example in the cryptographic scheme used for High Definition Content Protection (HDCP) (Crosby, Goldberg, Johnson, Song, & Wagner, 2001). The Blom's scheme has three main useful security features.

Firstly, it is mutually authenticating as both nodes must obtain their keying material from the same TA in order to obtain a common pairwise key. If a node receives

a message encrypted with their common pairwise key, the receiver can trust the authenticity of the sender. Secondly, it is a non-interactive scheme and there is no opportunity for an active adversary to participate and manipulate the scheme. The nodes only need to obtain some publicly available information about their counterparts and the TA is not required. Finally, it is unconditionally secure in the information theoretic sense in that, if no more than a certain number of nodes, λ ("capture threshold") are compromised, the scheme cannot be broken by an adversary with unlimited resources (Stinson, 2006) pp. 406. However, once the capture threshold is reached, the scheme can be completely broken very quickly. This was demonstrated in the cryptanalysis attack of the HDCP scheme (Crosby et al., 2001). To increase the capture threshold, the size of the keying material increases proportionally. Symmetric key establishment protocols do not scale well to large networks (Paar & Pelzl, 2010) p. 352, and this is true of Blom's.

One limitation in the Blom's scheme is that the pairwise key size is the same size as the data size used. The recommended key sizes by the US National Institute of Standards and Technology (NIST) (Barker, Barker, Burr, Polk, & Smid, 2012) p. 67, for the US Federal Government unclassified applications require key sizes of 112 bits or larger for protection to year 2030 and beyond. Large pairwise keys require more memory for storing the keying material.

The Blom's scheme would be useful for use in wireless sensor networks if it can be suitably modified. Various attempts have been made to modify the Blom's scheme to overcome its limitations. Some approaches use probabilistic ideas where nodes are given keying material from multiple keyspaces (S.-J. Wang, Tsai, & Chan, 2007). Another approach is to add some random perturbations to the keying material making it more difficult for the adversary to break the scheme (W. Zhang, Zhu, & Cao, 2007),(Yu, Lu, & Kuo, 2010). The question then, "Is it possible to modify the Blom's scheme using permutations of multiple keys such that it has security strengths of 128 bits or more,

given that any number of nodes can be compromised, and that the storage requirement for the keying material does not increase proportionally as the network size?"

1.2 Contributions

This thesis shows that it is possible to modify a symmetric scheme like the Blom's scheme so that it has adequate security strength, does not require large storage for the private keys, and is resilient against the compromise of tens of thousands of nodes.

The main idea is to have multiple keys and using them together in a single keyspace over the a small prime field $\mathbb{F}_p$. The TA has N master keys and assigns each node a set of η "public keys". These are used in permutations to obtain a set of $N\eta$ "private keys" which are stored in a random order at the node. This has the effect of breaking the links between the private keys with the public keys and the master keys used to compute them. These links are called the private-public-master-key associations (PPMka). If the adversary obtains the private keys, he must first discover the PPMka for each key before it can be used to attack the scheme. The PPMka information is unknown and ambiguous. By selecting suitable sizes of m, N, p, and η, the number of possible PPMka is so numerous that it requires an infeasibly large number of iterations, for example 2^{128} or more.

A pair of nodes would compute their pairwise "key-set" consisting of $N\eta^2$ integers using all permutations of the counterpart's public keys with its own private keys. The large number of integers computed enables large pairwise keys of 128 or more bits to be obtained. However, the key-set can become an avenue for an adversary to discover the PPMka. To prevent this, the prime modulus p is chosen to be small, for example, $p = 31$, so that the key-set has numerous identical integers, making it difficult to discover the PPMka. The adversary would have to capture tens of thousands of nodes to find a "traitor node", which when paired with another node, would expose

its PPMka. If a traitor node is found, the PPMka has a slightly better chance of being exposed in other nodes. The security of the scheme, and possible attacks, are analysed using mathematical tools including linear algebra, combinatorics, and probabilities. The analytical results are tested against computer simulated attacks to break the scheme.

1.2.1 The Proposed Scheme

The symmetric key establishment scheme proposed in this thesis, called the Blom-Yang key agreement (BYka) scheme, is able to achieve a security strength of 128 bits requiring 1170 bytes of memory for storing the private keys when implemented in the MICAz mote. The adversary cannot break the scheme even if tens of thousands of nodes are compromised. The implementation in the MICAz (Memsic Corp., n.d.) mote which has an ATmega128L processor running at 8 Mhz, is able to compute the pairwise keys in about 280 milliseconds. For a security strength of 80 bits, the computation time is only 104 milliseconds requiring 612 bytes of memory. For comparison, the scheme in (W. Zhang et al., 2007) using random perturbations based on the Blundo's bivariate symmetric key scheme (Blundo et al., 1995), was able to compute 80 bits keys in a time of 130 milliseconds. Public key cryptographic methods require much longer computation times. The fastest scheme using the identity-based scheme based on bilinear pairings in (L. B. Oliveira et al., 2011) was able to compute the pairings in 1.9 seconds, with 80 bits security.

Table (1.1) gives the shortest computation times for various security strengths using the proposed scheme implemented in the MICAz mote using the TinyOS code in Appendix B.4. The number n_c is the expected number of nodes required to find a traitor node, and Φ gives the probable number of iterations required to solve the N $(m \times m)$ system of equations to find the correct master keys. The number of bytes Q_o required for storing the private keys increases with security strength to about 1824 bytes for 192

Security Strength	n_c	Φ	$Q_o(B)$	$T_{comp}(ms)$	p	m	η	N
192	6.63×10^4	2^{193}	1824	342	61	38	4	12
128	1.38×10^7	2^{132}	1170	279	31	26	5	9
112	4.55×10^5	2^{114}	920	185	31	23	4	11
96	2.30×10^4	2^{97}	720	118	31	20	3	12
80	2.30×10^4	2^{83}	612	104	31	17	3	12
64	1.02×10^6	2^{64}	468	85	17	13	3	12

Table 1.1: Security Strength and Computation Times. n_c is the Traitor Capture Size, Φ is Number of Possible Solutions

bits security. The memory required for the computation algorithm is about 7000 bytes.

1.3 Security Model

The following defines the security model under which the proposed scheme operates and is evaluated. The security model comprises the system, the adversary, and the definition of system breakdown.

1.3.1 System Model

The system comprises three main components: the Trusted Authority, the sensor network comprising the nodes, and the operating environment.

Trusted Authority

All the nodes in the network belong to one administrative unit under the Trusted Authority (TA). The TA is responsible for generating and providing all the keying material. The TA can comprise one, or several entities. They can act jointly as a committee of TAs, or in a hierarchy with some acting as subsidiary TAs. We will refer to them collectively as TA. The TA is a secure entity and it is assumed that all

its secret information is protected against theft and leakage. The TA has access to a cryptographically secure random number generator to generate its keys.

Before a node is deployed, the TA would physically identify a node to verify its hardware and software, and then transmits the keying material to the node using a secure channel such as a direct cable connection. In this way, possession of valid keying material by a node proves that it is authentic. After the initial contact between the nodes and the TA, the TA is no longer required and plays no further part in the key establishment process.

Sensor Nodes

The sensor nodes have limited computing power, memory, and battery life, for example the MICAz mote has an 8-bit ATmega 8 MHz processor, 4 KB RAM, 4 KB EEPROM, and 128 KB Flash memory. They have access to secure cryptographic tools including the Advanced Encryption System (AES) algorithm, hash algorithms, and a pseudo random number generator (PRNG). The nodes are not provided with tamper proof hardware. It is assumed that if an adversary is able to physically take control of a node, all its data including secret keys in RAM and ROM can be extracted from the node either in the field or in the laboratory. The ease with which this can be done was demonstrated in (Hartung, Balasalle, & Han, 2005).

Deployment Space

The nodes may be installed in fixed physical locations, or are mobile and free to move about anywhere in the deployment space. The network may be ad hoc and the nodes are deployed as and when they are required. The number of nodes is large, up to tens of thousands belonging to the same TA. They communicate with their immediate neighbours using open radio protocols such as the IEEE 802.15.4 (IEEE, 2006) protocol. Their radio ranges are limited and the intermediate nodes along the way help to relay

messages to distant nodes.

1.3.2 Adversary Model

For schemes which require exchanges of multiple messages, adversary models such as that in (Dolev & Yao, 1983) would be useful. However, in our case, there is no interaction, except for obtaining the public *ID* of the target node. Our adversary model is thus simpler. In our model, the adversary is able to move freely anywhere in the deployment space to initiate conversations with any node, read unencrypted messages, replay, forge, block, and insert messages into the network. Overall, the adversary is a very powerful agent and, except for stealing the master keys from the TA, it is able to;

- capture and physically take control of any node,
- extract all data from the node's RAM and ROM memory,
- have access to powerful computing resources, and
- introduce new nodes into the target network space.

1.3.3 Security Outcomes

Definition 1 (Compromised node) *A node is compromised if an adversary is able to obtain all its keying material, for example, by physically taking control of the node and extracting the keys from ROM and RAM memory.*

Definition 2 (System Breakdown) *The system is considered broken, or compromised, if the adversary, by monitoring the transmissions and/or using keying material from compromised nodes, is able to:*

1. *compute the pairwise keys of any pair of uncompromised nodes, or*
2. *fabricate new valid keys for use in new nodes, or*
3. *compute the master keys of the Trusted Authority.*

Definition 3 (Security Strength) *This is a number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system, specified in bits and is a specific value from the set* $\{80, 112, 128, 192, 256\}$ *(Barker et al., 2012), p. 27.*

Exclusion – Identity Theft

If an adversary is able to steal the keying material from a node, it is able to use the information to create a new node (clone) with the same keys. Countermeasures against such identity thefts are not within the scope of this thesis.

1.4 Terminology

The following terms are in line with common usage as far as possible. Some terms have been defined loosely in the literature. While the exact definitions have varied as the subject developed, they do not affect the main concepts involved.

Key Establishment "Key establishment is a process or protocol whereby a shared secret becomes available to two or more parties, for subsequent cryptographic use." (A. J. Menezes, Oorschot, & Vanston, 2001) p.490.

In (Paar & Pelzl, 2010), key establishment schemes are classified into key transport and key agreement schemes.

Key Agreement Scheme "Key agreement scheme is a key establishment technique in which a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of these, (ideally) such that no party can predetermine the resulting value." (A. J. Menezes et al., 2001), p. 490.

Authenticated Key Establishment Scheme This "is a key establishment protocol in which one party is assured that no other party aside from a specifically identified second party (and possibly additional identified trusted parties) may gain access to a particular secret key." Adapted from (A. J. Menezes et al., 2001) p. 492.

Non-interactive Scheme "If two users only need to exchange their public keying material such as their *ID*'s and/or certificates, and this information is regarded as fixed, public information, it is regarded as a non-interactive scheme." (Stinson, 2006), p. 398.

Capture Threshold, λ This is the number of nodes that, if compromised, would break the Blom's scheme.

Unconditional Security "A cryptosystem is unconditionally or information-theoretically secure if it cannot be broken even with infinite computational resources." (Paar & Pelzl, 2010) p. 36.

Blom's Unconditional Security The Blom scheme is unconditionally secure against an adversary whose goal is to determine the secret pairwise key of a pair of uncompromised nodes, and can obtain at most $(\lambda - 1)$ compromised nodes. (Stinson, 2006) p. 399.

Secure Channel This is a physical (wireless) link for conveying messages between a pair of nodes, such that the messages are protected using cryptographic techniques so that an adversary cannot read the messages, the source of the messages can be assured, and the content can be proven to be intact. (A. J. Menezes et al., 2001), adapted.

Keying Material This it the set of data provided by the TA to each node for use in the key establishment process. It comprises the private keys, public keys, and the global

parameters.

Public Key This is a set of keying material unique to the node, provided by the TA, and is available to anyone in plain text.

Private Key This is a set of keying material, secret and unique to the node, provided by the TA and is never transmitted outside the node.

1.5 Structure of Thesis

The following Chapter 2 is a review of other work in sensor network key establishment. Chapter 3 presents the main concepts of the proposed scheme in this thesis. The security of the scheme is analysed in Chapter 4, showing that the weakness of the original Blom's scheme no longer applies by making the PPMka information unobtainable. Chapter 5 analyses how the PPMka may be discovered by analysing the probabilities of successfully obtaining the PPMka. Chapter 6 describes the experiments to verify the analytical results using a computer programme to implement the scheme, and simulating node capture to discover the PPMka. In addition, the scheme was also implemented in the MICAz mote to obtain some data on the computation times and resources required. Chapter 7 discusses how the practitioner may use the results to implement the scheme to achieve the desired security strengths and key computation times. The utility of the BYka scheme is demonstrated in the proposed single message authenticated protocol (SMAP). Chapter 8 gives conclusions including the strengths and weaknesses of the scheme and possible future work.

1.6 Publications

The following publications have been the result of this study:

Yang, M.L., Al-Anbuky, A., & Liu, W. (2012), A Fast and Efficient Key Agreement Scheme for Wireless Sensor Networks, *International Conference on Wireless and Mobile Communications, Venice. 24-29 June 2012*, pp. 231–237.

Yang, M.L., Al-Anbuky, A., & Liu, W. (2013), *The Multiple-Key Blom's Scheme for Key Establishment in Mobile Ad Hoc Sensor Networks*, The 19th Asia-Pacific Conference on Communications, Bali, Indonesia, 29-31 Aug 2013. pp. 422-427.

Yang, M.L., Al-Anbuky, A., & Liu, W. (2014), An Identity-Based Authentication Protocol for Sensor Networks, *IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore. 21-24 April 2014*

Yang, M.L., Al-Anbuky, A., & Liu, W. (2014), *Security of the Multiple-Key Blom's Key Agreement Scheme for Sensor Networks*, IFIP Advances in Information and Communication Technology, ICT Systems Security and Privacy Protection, Springer-Verlag GmBH, Berlin, Heideberg, pp. 66–79, June, 2014

Yang, M.L., Al-Anbuky, A., & Liu, W. (2014, June), *An Authenticated Key Agreement Scheme for Wireless Sensor Networks*, Journal of Sensors and Actuator Networks, 2014, 3(3), pp. 181-206; doi:10.3390/jsan3030181

1.7 Summary

Key establishment schemes are useful for nodes in large ad hoc mobile networks to compute their pairwise keys when needed. Symmetric key methods are most suitable for low resourced sensor devices but they tend to be of limited scalability. This thesis sets out to investigate whether such a scheme, the Blom's scheme, can be modified so that it can be secure for use in large networks and is resilient against node compromise. This is shown to be possible by using a new idea using multiple keys in permutations, random storage order, and operations over a small finite field, to render the private keys stolen from captured nodes unusable for breaking the scheme. This idea used in the

proposed key establishment scheme can attain a security strength of 128 bits and more, without the need for large memory, and is scalable for large networks.

Chapter 2

Literature Review

2.1 Introduction

Early works in wireless sensor network security focussed on developing efficient encryption schemes such as SNEP (Perrig, Szewczyk, Wen, Culler, & Tygar, 2002), TinySec (Karlof, Sastry, & Wagner, 2004), and MiniSec (Luk, Mezzour, Perrig, & Gilgor, 2007). Protocols were also developed for authenticating broadcast messages including TESLA (Perrig, Canetti, Tygar, & Song, 2002), μTESLA (Fan, Chen, & Eltoweissy, 2005), and (J. Zhang, Yu, & Liu, 2009). These used global network keys which are undesirable as, if stolen, the whole network would be compromised. Later works began to address the need for pairwise keys, initially using symmetric keys, first using pre-distribution, and later using key agreement methods. At the same time, with the growing use of Public Key Cryptographic (PKC) algorithms for securing communications in computer networks, these PKC algorithms began to be investigated for use in low resourced devices as well. As PKC techniques require additional mechanisms for authenticating the public keys, more recent works started to use Identity-Based Cryptographic (IBC) schemes. This chapter focusses on work in authenticated pairwise key establishment schemes for sensor networks.

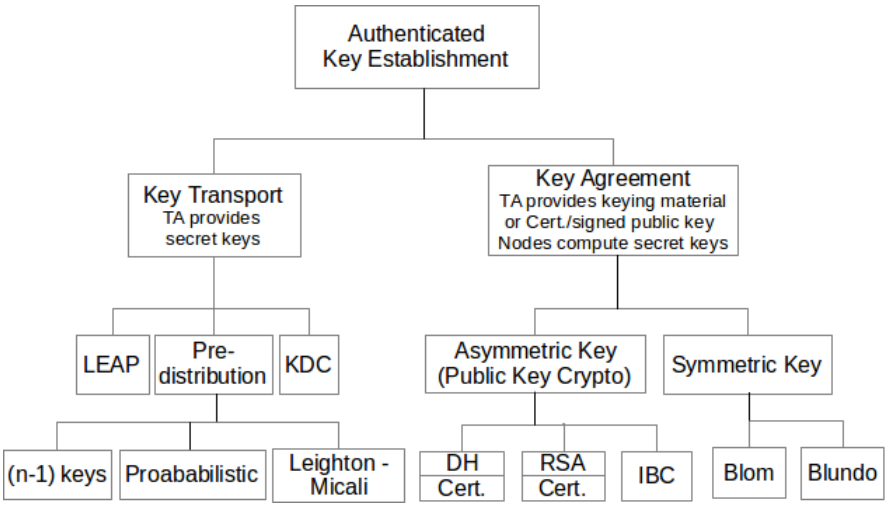


Figure 2.1: Classification of Authenticated Key Establishment Schemes

Key establishment schemes enable pairs of nodes to share a common pairwise key when they need to send encrypted messages to each other. Several ways to classify key establishment schemes have been observed in the literature as the subject has developed. Some are based on the type of cryptographic algorithms used, whether the participation of a third party is required, etc. The classification used in (Paar & Pelzl, 2010) p. 331, divides them into two main methods: key transport methods and key agreement methods. In key transport methods, one party, the TA, generates and distributes all the secret keys to the nodes. On the other hand, in key agreement methods, the parties jointly generate their secret pairwise key. For authentication, the TA is required to provide certificates, signed public keys, authentication keys, or the keying material for the nodes to compute their secret pairwise keys. The cryptographic system may be based on symmetric key or public key cryptosystems. The classification used in this thesis is shown in Fig. (2.1).

2.2 Key Transport Schemes

Most of the symmetric key establishment schemes are key transport schemes. The computations can involve hash functions to protect the keys or to derive keys from

keying material provided by the TA. The TA can also provide all the pairwise keys that are needed. The nodes obtain their pairwise keys using one of three methods: pre-distribution, probabilistic, or key distribution centres.

2.2.1 Pre-distribution Schemes

The key pre-distribution method is one of the simplest methods of distributing pairwise keys if the number of nodes is small. The TA generates all the pairwise keys that are needed. If the number of nodes is n, there are $\frac{n(n-1)}{2}$ pairwise keys. Each node is provided with $(n-1)$ keys so that it is able to share a common key with every other node. The scheme is very easy to implement requiring just lookup operations. However, the memory requirement increases proportionally with the network size. Also, new nodes cannot be added since the nodes already deployed do not have the new keys.

LEAP In the Local Encryption and Authentication Protocol (LEAP) (Zhu, Setia, & Jajodia, 2003) the nodes are provided with the global key for authentication. As nodes are not provided with tamper proof mechanisms to keep the cost low, the nodes delete the global authentication key within a time window after deployment, assuming that it cannot be stolen by an adversary during this period. The nodes use the global key called the "initial key", K_I to derive their own secret "master key". For example, a node with *ID* u derives its master key $K_u = f_{K_I}(u)$, where f is a pseudo random number generator. After deployment, it broadcasts u, its *ID*. The neighbour node responds with its *ID* v, and an acknowledgement encrypted with its own master key K_v. Node u has K_I and can compute the neighbour's master key $K_v = f_{K_I}(v)$ which it uses to decrypt the message containing the *ID*'s u, v. If successful, node u authenticates node v and obtains the pairwise key $K_{uv} = f_{K_v}(u)$. Node v computes K_{uv} in the same way. The nodes u and v are now "secured" using their pairwise key.

While node u has the key K_I, it can compute the master key of any node i using

$K_i = f_{K_I}(i)$ and their pairwise key $K_{ui} = f_{K_i}(i)$. Node i trusts that only a node which has K_I can compute its master and hence their pairwise key. After obtaining the pairwise keys with its neighbours, the master keys are deleted. If the global key K_I is stolen, the entire network is compromised. To minimise this risk, the global initial key is deleted within a certain time after deployment. To further minimise this risk, the LEAP+ protocol broadcast the *ID*s of new nodes after the time window, for example using $\mu TESLA$ (Perrig, Canetti, et al., 2002) authenticated broadcast protocol, so that nodes with these *ID*s are no longer valid.

The LEAP protocol is suitable for fixed topologies where a new node uses the global authentication key to authenticate other nodes which are already authenticated and secured on the network. After authenticating the secured nodes, it becomes a secured node, joins the network and deletes the initial key. After this, it can only be authenticated by a newly deployed node, but not by nodes which are already secured in the network since it no longer has the initial key K_I if it moves to another part of the network space. It is not suitable for ad hoc mobile topologies.

Probabilistic Schemes For a fixed topology, it is not necessary for a node to store all the possible pairwise keys since it has only a few immediate neighbours. Therefore, prior to deployment, a node can be deployed with a small number of keys which hopefully, would have a common key with the neighbour's pool. This was pioneered in the scheme by (Eschenauer & Gligor, 2002) where nodes are given subsets of keys from the global key pool. After deployment, pairs of nodes run a protocol to discover their shared keys to establish secure links. If a pair of nodes does not have a shared key, they may still be able to establish a secure link using secured mutual intermediary nodes. If one node becomes compromised and its keys stolen, only part of the network is affected.

To improve on the probability of pairs of nodes sharing keys in their key pool, the

scheme in (Camtepe, Seyit Ahmet and Yener, Bülent, 2007) used combinatorial block design theory to build the key distribution scheme. It is deterministic and also includes probabilistic extensions. Each sensor node has a key-chain of keys selected from a pool. The nodes use this to discover their common keys, and if there are no shared keys, intermediary nodes are used to help establish secure links.

Leighton-Macali's Scheme

In this scheme (Leighton & Micali, 1994), the TA provides all the nodes with a secret key. It computes the hash of all the pairs of secret keys, and stores them in a database. This database can be public and if the hash algorithm is strong, an adversary cannot obtain the keys. A node is able to compute a shared key with another node by using its secret key with the appropriate entry in the database. In this way a node only needs to store its secret key if it has access to the database. In the scheme the TA uses two secret master keys, K and K', to create two private keys for each node by hashing its master key with the node's *ID*. For example for node i, its keys are $K_i = h(K, i)$, and the authentication key $K'_i = h(K', i)$. It also publishes two databases matrix, $P_{i,j} = h(K_i, j) \oplus h(K_j, i)$, and $A_{i,j} = h(K'_i, h(K_j, i))$. A node *i* can derive a pairwise secret with node *j* by looking up the database and computing $K_{i,j} = P_{i,j} \oplus h(K_i, j) = h(K_j, i)$. Node *j* can also compute $K_{i,j} = h(K_j, i)$.

For authentication, node *i* checks that $h(K'_i, K_{i,j}) = A_{i,j}$ and verifies the authenticity of the key $K_{i,j}$. Thus a node can derive an authentic pairwise secret with another node by accessing the public databases, $P_{i,j}$ and $A_{i,j}$. Alternatively, each node *i* stores the i^{th} row of these matrices which has $n - 1$ entries. This would require a large amount of memory if there is a large number of nodes. This scheme is, in effect, a method of distributing all the possible pairwise keys using a publicly available database.

The security of the scheme depends on the strength of the one-way hash functions. It is very efficient and requires simple computations. It is also very scalable and to add

new nodes, only the databases need to be updated. The main drawback is that the public databases must be accessible for full connectivity which is difficult for ad hoc networks. This may be mitigated by installing the databases in trusted key-server nodes distributed throughout the network. Alternatively, each node can store its own copy of the public key database but this would need a large memory and cannot accommodate new nodes. The probabilistic idea from Eschenauer and Gilgor was also used in (M. Liu et al., 2009) where the nodes were preloaded with some of the public key databases. In addition, key-server nodes were distributed throughout the network to provide the keys if they were not in the node's memory.

PIKE

In the peer intermediaries for key establishment scheme (PIKE) (Chan & Perrig, 2005) , the basic idea is that the sensor nodes are placed in a virtual square grid. Each node at (x, y) has pairwise keys with the other nodes in the same row and column. Nodes then need to store a total of only $2(\sqrt{n} - 1)$ keys. If a node needs to obtain a pairwise key with some other node that does not lie on the same row or column, it needs to find an intermediary node which lies on the intersection of their coordinates. If this intermediary is contactable by both nodes, i.e. also located in their physical vicinity, it can be used to authenticate and establish their pairwise keys. If not, another intermediary needs to be found. This may not be possible in mobile ad hoc networks. Key establishment would also involve additional communications through the intermediary nodes. The scheme can be extended to three or more dimensions to reduce the amount of communications needed to find a shared intermediary.

2.2.2 Key Distribution Centers

In this method, a central entity called the Key Distribution Center (KDC) is responsible for providing the pairwise keys to the nodes. When a node wishes to communicate with another node, it contacts the KDC which, after authenticating the node, will generate and provide the required pairwise key. To authenticate itself to the KDC, each node is provided with a secret key shared with the KDC. In this way, a node needs to store only one key and still be able to communicate with any other node using its KDC provided pairwise key. This approach is used in the Kerberos protocol (Steiner, Neuman, & Schiller, 1988) widely used in computer networks.

Key Distribution Centres (KDC) is of limited use to ad hoc mobile WSN as the KDC has to be reachable at all times and there is a concentration of traffic in the intermediate nodes close to the KDC. In the Zigbee protocol (ZigBee, 2005) there is provision for the Trust Centre (TC) to act like the KDC for the nodes in the cluster. There is hardly any other literature on KDC in WSN, except in a comparison between using a lightweight version of Kerberos against the ECDH/ECMQV (Grosschadl et al., 2007) key agreement method. Here, while the energy used in the Kerberos protocol is comparatively very low if the KDC is an immediate neighbour, using only about 47.6 mJ compared to the ECMQV's 84.6 kJ, most of the energy is spent in the radio communications. If the number of intermediary nodes is more than 2 as would be the case of large networks, then ECMQV is more efficient. In ad hoc mobile networks, this scheme is of limited use due to the requirement for the trusted centre to be online and reachable at all times.

2.3 Key Agreement Schemes

These schemes enable a pair of nodes to compute their pairwise keys after obtaining some public information about their counterparts. Key agreement schemes are most useful for mobile ad hoc networks. Here, pairs of nodes compute a common secret key without the participation of a third party. In schemes with authentication, the Trusted Authority is only needed to compute and pre-install the keying material in each node prior to deployment. The keying material is unique to each node and if a node is compromised, the impact on the network is limited. The key agreement process begins with the nodes exchanging some keying material (called "public key") over the insecure channel. Any other node is able to intercept and read the public key. Each node then uses the counterpart's public key with its own secret keying material (called "private key") to derive a common secret key. An adversary, having monitored the exchanges, must not be able to compute the common secret key.

Two classes of key agreement schemes are available: one is based on asymmetric key cryptographic, or commonly known as Public Key Cryptography (PKC), and the other is based on symmetric key cryptographic methods.

2.3.1 Public Key Cryptographic Schemes

Key agreement schemes using Public Key Cryptography (PKC) primitives such as the Diffie-Hellman (Diffie & Hellman, 1976) and Shamir & Adleman (RSA)(Rivest et al., 1978) algorithms are already widely used in computer networks. The security of these schemes is based on the intractability of known mathematical problems such as the factorisation of large integers and the Discrete Logarithm Problem (DLP) (Paar & Pelzl, 2010) p. 217. Consequently, to attain the required security strength, they need to use large integers of 1024 bits or larger for operations over the prime field. With Elliptic Curve Cryptography (ECC), 160 bits or more are required (Barker et al., 2012). There

has been much interest in adapting PKC schemes for sensor networks. The main challenges are the cost of the complex computations, memory, and energy resources required when implemented in low resourced sensor devices.

Diffie-Hellman Algorithm In the Diffie-Hellman algorithm, a pair of nodes A and B are able to compute a secret key after obtaining each other's public key. To do this, they use known public parameters; the generator g and the cyclic group of prime order p, $\mathbb{Z}_p^*$. For example, node A selects a random secret number $a \in \mathbb{Z}_q^*$, computes its public key $g^a \pmod p$ and sends it to node B over the insecure channel. Similarly B selects a random $b \in \mathbb{Z}_q^*$, computes $g^b \pmod p$ and sends it to A. Node A computes $K_{AB} = (g^a)^b \pmod p$, and node B computes $K_{BA} = (g^b)^a \pmod p = K_{AB}$. It is assumed that there is no efficient algorithm for an eavesdropping adversary, knowing g^a and g^b to compute g^{ab} (the Computational Diffie-Hellman (CDH) problem). In addition, it is assumed that given g^a and g^b, for $a, b, c \in \mathbb{Z}_q^*$ there is no efficient algorithm to distinguish between the triplets $\langle g^a, g^b, g^{ab} \rangle$ and $\langle g^a, g^b, g^c \rangle$ (Boneh, 1998).

The basic scheme is vulnerable to the man-in-the-middle (MITM) attack. Any node is able to generate its own public-private key pair. An adversary E can interpose itself between A and B. It can impersonate B and forward its own public key K_E to A. If A accepts that public key K_E belongs to B and proceeds to compute the pairwise key, it will obtain the pairwise key K_{AE} with E instead of B. The adversary similarly impersonates A by sending its public key K_E which B will use to compute the pairwise key $K_B E$. The adversary can intercept and decrypt messages between A and B and forward them without their knowledge. To mitigate this MITM attack, some other mechanism must be used so that the nodes can authenticate each other's public keys.

Symmetric Key	RSA and DH	Elliptic Curve
80	1024	160
112	2048	224
128	3072	256

Table 2.1: NIST Recommended Key Sizes in bits

El Gamal Algorithm The El Gamal algorithm (Elgamal, 1985) is based on the DH method, but allows users to encrypt messages using their public keys. The node A chooses a large prime p_A, an integer α_A modulo p_A, and a random private key x. It computes $\beta_A = \alpha_A^x \pmod{p_A}$, and publishes the public key $\langle p_A, \alpha_A, \beta_A \rangle$. Another node B can encrypt a message M to A as follows:

Node B chooses a random secret k, computes $r \equiv \alpha_A^k \pmod{p_A}$, and $t = \beta_A^k M \pmod{p_A}$ and discards k. B sends $\langle r, t \rangle$ to A who decrypts, $tr^{-x} \equiv \beta_A^k M(\alpha_A^k)^{-x} \equiv (\alpha_A^x)^k M(\alpha_A^k)^{-x} \equiv M \pmod{p_A}$.

As in the DH scheme since any node can generate its own public and private keys, it is open to the MITM attack and another mechanism is required to authenticate the public keys.

ECDH For WSN, algorithms based on Elliptic Curve Cryptographic (ECC) are attractive due to it having less demand on resources compared to other PKC methods. It is considered that ECC using 160 bits is comparable in security to RSA and DH using 1024 bits according to (National Security Agency, 2009), see Table 2.1. Most later work on PKC in WSN uses ECC algorithms as it is more efficient for the same security strength. Many researches including (H. Wang & Li, 2006),(Ugus et al., 2007),(Grosschadl et al., 2007) and (Lederer et al., 2009) are aimed at making the ECC computations more efficient using various optimization techniques.

RSA Methods The Rivest, Shamir & Adleman (RSA) (Rivest et al., 1978) algorithm can be used to transport a secret key to another node over an insecure channel. The security strength is based on the difficulty of factoring a large number into two large primes. It works as follows: The node A chooses two distinct primes p and q, computes $n = pq$ and then computes the totient function $\phi(n) = (p-1)(q-1)$. The primes p and q are then deleted. It then selects a private key e and a public key d such that $d \cdot e \equiv 1 \pmod{\phi(n)}$ and e and $\phi(n)$ are relatively prime, i.e. $gcd(e, \phi(n)) = 1$. The public key $\langle e, n \rangle$ is published.

Euler's totient function ϕ is the number of integers in the ring $\mathbb{Z}_m$, i.e. the set of integers $\{0, 1, \ldots, m-1\}$, that is relatively prime to m.

Another node is able to encrypt a message containing a secret key M to A by computing and sending the cipher text $C = M^e \pmod{n}$. Node A is able to decrypt the message by computing $C^d \pmod{n} \equiv (M^e)^d \equiv M^{k\phi(n)+1} \pmod{n}$, where k is some integer. Using the identity due to Euler and Fermat, $M^{\phi(n)} \equiv 1 \pmod{n}$, $C^d \equiv M$ and node B obtains the secret key in the message.

The scheme is also vulnerable to the MITM attack as any node can generate its own private and public keys. An additional mechanism to authenticate the public key is required, such as having the public key "signed" by the Trusted Authority. In this case, the TA has a private key d_T, and installs the node with the public key $\langle e_T, p_T \rangle$. It obtains A's public key $\langle e, n \rangle$ and encrypts e in message $M = e^{d_T}$. When node B wishes to encrypt a secret key to A, it first obtains M and computes M^{e_T} to obtain the public key e. Then using this public key authenticated by the TA, node B uses e to encrypt a secret key to A.

Challenges for Sensor Nodes

The PKC algorithms involve complex mathematical operations including exponentiations of integers 1024 bits or larger over the finite field, or point operations involving

160 bits or larger on elliptic curves. The main challenges for sensor networks are the limited resources available in the sensor nodes. The low computational power often means the computations take a considerable amount of time and energy, not only for encryption and decryption but also for verification of the counterpart's public keys. For instance, in (Gura, Patel, Wander, Eberle, & Shantz, 2004) the RSA-1024 implementation in an 8 MHz ATmega128 processor took 0.43 seconds for the public key operations (signature verification) while the slower private key operations took 10.99 seconds. The work also compared between RSA and ECC algorithms using 160 bits for ECC equivalent to RSA-1024. They showed that for ECC-160, private-key operations (signature generation) took 0.81 seconds which is about 10 times faster than RSA-1024. For public-key operations, ECC-160 took 0.81 seconds, about two times slower than RSA-1024. The code sizes were also substantially larger for RSA-1024 requiring 1,073 bytes (public key), 6,292 bytes (private key) compared to ECC-160 requiring 3,682 bytes.

Implementation Using RSA, TinyPK In the key agreement scheme using RSA-1024, TinyPK (Watro et al., 2004), nodes used the RSA techniques to authenticate each other and derive their session key. The memory requirements were reported to be 12.4 KB ROM and 1.167 KB RAM in the MICA2 platform. The public key operation took 14.5 seconds. There was no reported execution time for the private-key operations but they reported that it was extrapolated to be tens of minutes.

TinyECC A suite of code, TinyECC (A. Liu & Ning, 2008) (M. Liu et al., 2009) was developed for use with TinyOS (Levis et al., 2005), the operating system developed for sensor devices. The key agreement was done using elliptic curve Diffie-Hellman (ECDH) and public key authentication using the Elliptic Curve Digital Signature Algorithm (ECDSA). The TinyECC library can be configured for optimisations for large

integer operations, and optimizations for ECC point operations. Their results on the MICAz mote with all optimisations turned on, showed that the whole process including signature verification and ECDH key agreement can take a total of 6.2 seconds excluding initialisation times of about 5.2 seconds. The RAM required was in the order of 1.5 KB. With all optimizations turned off, it took more than 120 seconds for signing, verification and key establishment using ECDH.

Public Key Authentication A fundamental problem for PKC is that the public keys must be authenticated using a common trusted entity. This can be done by having the TA provide a certificate with the signed public key.

Transmission of the signatures or certificates involves a large number of bits consuming a substantial amount of energy. For example in the ECHD-ECDSA protocol implemented in (de Meulenaer, Gosset, Standaert, & Pereira, 2008) 2,208 bits were communicated in the process and the radio communications energy made up 17% of the total 236 mJ of energy used in the whole key agreement process.

Preloaded Public Keys To side-step this verification process, the public key of the counterpart node may be preloaded. This approach is taken in the elliptic curve Menezes Qu Vanstone (ECMQV) algorithm (Law, Menezes, Qu, Solinas, & Vanstone, 2003) where nodes have been pre-installed with the neighbour's static public key signed by the TA. This is used to authenticate itself to the neighbour and to compute a secret key after exchanging an ephemeral public key. However, this method is not practical for mobile ad hoc networks as the topology is not known prior.

Merkle Tree for Public Key Authentication If the node is preloaded by the TA with the public keys of other nodes, then there is no need to separately authenticate them. Alternatively, to save on storing all the public keys, in (Du, Wang, & Ning, 2005),

the nodes use a hash tree, called a Merkle tree to authenticate the neighbour's public key. Prior to deployment, a node is placed on the Merkle tree and is given an *ID*, its public key, and "proofs" which are hash values of its sibling nodes, its parent's siblings, grandparent's siblings, etc., up to, but not including the root. It also has the root's hash. For a node A to authenticate its public key to node B, it sends to B its *ID*, its public key and proofs of the nodes along the path to the root. Node B would compute the hash of ID_A and the public key, and all the proofs along the way. Finally it will obtain the root hash and if this compares correctly to its stored root hash, the public key is accepted. In this way each node only needs to store a number of hashes equal to the number of levels of the tree, the root hash, and its own *ID* and public key. If there are n nodes, the number of levels is $log_2(n)$. In (Du et al., 2005), the amount of storage and communications was further reduced by using pre-deployment knowledge of the network topology to trim the Merkle tree into many smaller trees.

2.3.2 Identity-Based Cryptography

The major hurdle of public key authentication in PKC techniques can be circumvented by using Identity-Based Cryptography (IBC). The idea was first conceptualised by Shamir (Shamir, 1985) in which node A is able to encrypt a message to node B whose public key is formed from its *ID* such as its email address, node name, etc. The sender node A, using the TA's "public parameters" (similar to public key) and B's *ID* would compute B's public key and use it to encrypt messages to B. Only B can decrypt the message using its own matching private key provided by the TA. The TA does not participate in the exchange after providing the private keys and its own public parameters to the nodes. This is an ideal mechanism for ad hoc mobile networks, also argued for in (L. B. Oliveira et al., 2007), and most works using PKC for sensor networks are actively pursuing this approach. Interestingly, in (A. J. Menezes et al.,

2001), p. 538, it was pointed out that "Blom was apparently the first to propose an identity-based (or more accurately, index-based) key establishment protocol". However, identity-based cryptosystems currently refer to those using PKC algorithms.

Boneh and Franklin Schemes

A concrete implementation of such a scheme was proposed by Boneh and Franklin (Boneh & Franklin, 2001) using bilinear mapping based on the Weil Pairing. An implementation was done by Cheng et al. (Cheng, Yang, Wang, & Huang, 2006).

TinyPBC The implementation called TinyTate (L. B. Oliveira et al., 2007) and later developed into TinyPBC used the Tate Pairing (L. Oliveira, Scott, Lopez, & Dahab, 2008) over supersingular binary curves. This implementation was able to compute the pairing, the most expensive operation in PBC, in only 5.5 seconds using the ATmega128 processor requiring 47.9 KB ROM, and 368 bytes of RAM of which 2.867 KB was in the stack.

This was followed by the work in (Szczechowiak & Collier, 2009), also based on the Tate pairing implemented on 8-bit sensor devices like the MICA2 using the ATMega128 processor. In this work, the pairing took 2.66 second. The required ROM was 47.41 KB, and RAM was 3.17 KB, using up almost all the mote's RAM, although most were in stack memory and can be reclaimed after the computation completes.

A later work (L. B. Oliveira et al., 2011) implemented in the ATmega128L processor improved the pairing time to just 1.9 seconds. The memory required was 37.9 KB ROM, and 3.6 KB RAM of which 3.1 KB were in the stack. The security strength was only 80 bits which is now considered too small according to NIST (Barker et al., 2012).

ID-Based DH The method in (Fiore & Gennaro, 2010) is an authenticated protocol that does not require bilinear maps but uses the Diffle-Hellman protocol in a manner

inspired by the Menezes-Qu-Vanstone (MQV) (Law et al., 2003). For authentication, it uses the Schnorr's signature (Schnorr, n.d.) created by the TA from the *ID*. The operations are over elliptic curves with equivalent 128 bit security strength. To initiate the key agreement process, only 512 bits need to be exchanged.

Security of IBC Schemes

The security of the bilinear pairing schemes depends on the hardness of the Discrete Logarithm Problem (DLP) in $\mathbb{F}_{q^k}$ and k must be large enough, at least 1024 bits, to achieve 80-bit security strength. For 128 bit security, 1536 bits would be required. In (Szczechowiak & Collier, 2009), it is believed that the security was roughly the same as that of the DLP over $\mathbb{F}_{2^m}$ where $m \approx 1084$. Pairing based IBC schemes work with 1024 bit numbers for 80-bit security level (equivalent to RSA-1024).

To date, pairing based cryptography of 923 bits was reportedly broken in 148.2 days using 21 servers (252 cores) (Hayashi, Shimoyama, Shinohara, & Takagi, 2012) even though researchers have expressed that such cryptographic systems greater than 900 bit will take thousands of years to crack. As more research into efficient procedures and better, faster and more computing resources are available, these PKC methods would require a larger number of bits to remain secure.

2.3.3 Symmetric-key Key Agreement Schemes

Another type of key agreement schemes, usually referred to as symmetric-key schemes, requires pairs of nodes to obtain some public information from their counterparts to compute a pairwise key. The public information is often referred to as their "public keys" even though a more accurate term may be "public keying material". These are often just an integer which serves as the node's *ID*. It is tempting to call them identity-based schemes but as these do not use PKC techniques, this is not usually done in the literature.

These symmetric schemes use simple modulo arithmetic operations without the need for large integers and exponentiations. Consequently, they are computationally less intensive and are fast and efficient. However, the main limitation is the size of the keying material which increases as the network size. Only two main schemes were reported: Blom's scheme and Blundo's polynomial scheme. In all these schemes, the TA generates all the keys for the nodes. Each set of keys consists of two parts, one part is secret and unique (private), and the other can be exchanged with another node publicly over the insecure channel. The nodes use their own private keys with the counterpart's public keys to compute their common pairwise keys.

Blom's Scheme

The earliest scheme was by Blom (Blom, 1983), (Blom, 1984) though its limitation for practical application was also quickly pointed out. The Blom's scheme uses a master key which is a symmetric $(m \times m)$ matrix. It is unconditionally secure if less than m nodes are captured (Stinson, 2006) pp. 406. Assuming that nodes are not tamper proof, the size of the network is then limited in that a large m requires proportionally large memory for storing the private keys. One method to overcome this in (Du, Han, Deng, & Varshney, 2003) used multiple small size key spaces. Using the probabilistic idea similar to Eschenauer-Gilgor's, the scheme enables pairs of nodes sharing common key spaces to form pairwise links using one of the common key spaces. In this scheme, the aim is to achieve full connectivity, but not necessarily complete connectivity like a full mesh.

An implementation using this scheme was the multi-party conference key agreement protocol in (H.-S. Lee, Lee, & Lee, 2003). This modification to the scheme allows multiple nodes to derive a common secret key. In another, the modified Blom's scheme in (J. Lee & Stinson, 2005) used multiple key spaces in the network graph for better resiliency against node capture. It improves the resilience as there is a smaller probability

of capturing all the required nodes from the same key space.

The number of nodes in a fully secure network can be increased by using multiple key spaces. In (Du et al., 2003), ω key spaces are generated and each node is given a sub-set of τ randomly chosen keys from ω. After deployment, nodes discover their common keys and use the Blom's scheme to form pairwise keys. With more key spaces, more storage is required. Each node would need $m \times \tau \times b$ bits storage for the private vectors. The scheme uses a similar idea to the probabilistic scheme of Eschenaeur-Gligor (Eschenauer & Gligor, 2002) where nodes are given a random set of keys from a global key space. In these schemes the aim is to achieve full connectivity, but not necessarily complete connectivity like a full mesh. A similar approach also uses Blom's scheme with multiple key spaces to improve resistance to the Sybil attack (S.-J. Wang et al., 2007).

In (Chen, Yao, & Wen, 2008), a scheme for a clustered topology was proposed. Here, the cluster-heads implement the Blom's scheme to derive pairwise keys among themselves. Non cluster-head nodes do not implement the Blom's scheme. Prior to deployment, the base station computes the pairwise keys of this node with a certain number of associated cluster-heads. These are then combined into a secret key K_i and stored in the node, together with the identities (*ID*s) of the associated cluster-heads. When a node needs to establish a secure link with a physical cluster-head, it sends its own *ID* and the *ID*s of its associated cluster-heads. The physical cluster-head forwards the node's *ID* to the associated cluster-heads to compute the pairwise keys using Blom's scheme and thereby derives the secret key. In this way, non-cluster head nodes store minimum keying material and do not need to perform any key computation. Instead, these are delegated to the cluster heads which carry the additional load of communicating with other cluster heads to derive the key with a non cluster-head node. The network size would still be limited to the $(m - 1)$ cluster head nodes for a fully secure network. Since cluster-heads establish pairwise keys among themselves using

the basic Blom's scheme, the key size and memory requirements, and the number of cluster heads would still be limited by the original scheme.

A scheme in which the private vectors of the nodes can be updated was proposed in (Zhou & He, 2009). In this scheme, the modified Blom's scheme used hashed values of the prime seeds and the nodes have private vectors which are hashes of the original private vectors. The implementation used a protocol for nodes to update their private vectors and pairwise keys.

In (Chien, Chen, & Shen, 2008), a mixture of the Blom's scheme with the KDC scheme is used. The Blom's scheme is protected by deleting the private keys once the pairwise keys are established with the neighbours. To cater for a new node i joining the network, in addition to the public and private keys for the Blom's scheme, it is provided with a global secret random number R_s (effectively a global secret key) and another secret key shared with the base station only, $K_{BS,i}$. If node A wishes to establish a pairwise key with node B, it uses R_s to encrypt a random number R_a and sends it to node B, and vice versa. After obtaining the pairwise key $K_{a,b}$ using the Blom's scheme, the random numbers are used with it to obtain their session key $EK_{a,b} = H(K_{AB} \cdot R_a \cdot R_b)$. Once this is completed, all the keying material except for the pairwise key $EK_{a,b}$ and the secret key shared with the base station, for example $K_{BS,a}$ in node A, are erased. If the node is captured, the private keys are unavailable and the Blom's scheme cannot be attacked. New nodes added to the network do not implement Blom's scheme to establish pairwise keys with existing nodes. Instead, a new node u is deployed with two secret keys K_u and $K_{BS,u}$ which the neighbour node uses to authenticate node u with the base station, and then establishes their pairwise key. The key sizes obtained are not reported and it would require large integer operations to obtain large keys.

The implementation of Blom's scheme in (Yu et al., 2010), also uses random perturbations. Here, small constrained random perturbations are added to the private

keys to break the direct connection to the master key, making it more difficult to break. The pairwise keys computed are identical after the effect of the small random perturbations are removed. To obtain large pairwise keys, multiple separate instances ξ of the Blom's scheme are run to obtain ξ pairwise keys and concatenated together to obtain a large pairwise key of 128 bits.

Blundo's Polynomial Scheme

The Blundo's scheme (Blundo et al., 1995), inspired by the Blom's scheme, enables a group of nodes to derive a common "conference" key. The Blom's scheme is a special case where the the group size is 2. In this scheme, the TA generates a symmetric polynomial in t-variables of degree k. Each user with *ID* i is given a share of the polynomial evaluated at i. If a group of users with *ID*s $(i_1, i_2, .., i_t)$ want to form a conference key, each user evaluates the polynomial using the counterparts' *ID*s and obtains a common key. This scheme is unconditionally secure if less than k users collude. The Blundo's scheme for two parties uses symmetric bi-variate polynomials and is equivalent to the Blom's scheme.

(D. Liu, Ning, & Li, 2003) implemented the Blundo's scheme using a bi-variate polynomial combined with a probabilistic method inspired by the approaches used in (Eschenauer & Gligor, 2002) and (Chan, Perrig, & Song, 2003). The TA generates a pool of bi-variate polynomials and assigns each node a subset from the pool. Since the possibility of pairs of nodes not sharing any common set can occur, the scheme has an additional step of using other mutual intermediary nodes to form pairwise keys. The key size is still limited to the word size of the system in this case using 64 bits in MICAz motes.

A similar implementation of the Blundo scheme was done in (Zheng & Dai, 2008), combined with a probabilistic approach similar to the q-composite scheme in (Chan & Perrig, 2005) with each node having a pool of polynomials in order to increase the

resilience. This scheme can achieve almost 100% connectivity with reduced storage requirement.

A different approach in (W. Zhang et al., 2007), added random perturbations to the share of the bivariate polynomial evaluated using the node's ID. This makes it harder to break the symmetric polynomials if these shares are captured. To obtain large pairwise keys, they use multiple separate instances of symmetric polynomials with the node's *ID* to obtain segments of pairwise keys which are combined to form large pairwise keys. This scheme was able to obtain 80 bits keys efficiently in a time of about 130 milliseconds, requiring about 15 KB ROM, and 0.33 KB RAM in MICA2 with ATmega128L processor, 128 KB flash memory.

2.4 Summary

Pairwise key establishment can be achieved by either the key transport or key agreement methods. The key transport methods are not suitable for mobile hoc networks as they would need a large memory to store all the keys. Alternatively, using key servers or key databases requires connectivity which may not be available. It is also possible to use secured intermediary nodes to help with key establishment but full connectivity is not assured. Key agreement methods are most suitable for ad hoc mobile networks which have no pre-determined network topology, and there is no need for a third party to take part in the pairwise key derivation process.

Many key agreement schemes based on public key cryptographic algorithms are already widely used for computer networks. These schemes can be used for very large networks without the nodes requiring more storage. The algorithms use complex mathematical operations on large integers. A lot of work has been done to adapt them for use in sensor devices but the computations require substantial computing, memory, energy resources, and take considerable time often measured in minutes. The

public keys need to be authenticated requiring an exchange of a substantial number of bits, consuming energy for radio transmission. Another approach, the identity-based schemes based on bilinear pairing have also been studied for sensor networks as they have implicit authentication. It is also based on public key cryptographic algorithms. Currently the best identity-based scheme using bilinear pairing takes 1.9 seconds, achieving 80-bit security strength. The security of these PKC schemes are based on complex operations on large integers and their suitability for low resourced devices would be a continual challenge since the underlying security basis; the hardness of the DLP and the factorisation of large numbers, will continually be challenged with better and faster algorithms and hardware. Operations using larger and larger integer sizes will be required to remain secure.

Key agreement schemes using symmetric key cryptography are often not considered identity-based even though they are "index-based". These include the schemes by Blom and Blundo. They use simple mathematical modulo multiplication and addition operations. However they are not easily scalable for large networks. If a certain number of nodes, the capture threshold, is compromised, the entire scheme is broken. To increase the capture threshold, the size of the private keys also increases proportionally. The Blom's method including Blundo's scheme (which is in fact identical to the Blom's scheme for the two-party case), is useful for sensor networks if the limitation due to the capture threshold can be overcome. Some attempts have been made in this direction using multiple key-spaces and random perturbations to make it more difficult for the adversary to use the captured private keys to derive the master keys. Some schemes also use multiple instances of the Blom's method to obtain a large pairwise key size. What has not been studied is the possibility of using multiple keys, not separately, but in a single instance with permutations over a small prime field, so that the connections of the private keys to the master keys and public keys are broken and cannot be recovered easily. This will render the captured keys unusable for attacking the scheme, allowing

it to break free of the capture threshold without the proportional increase of storage requirements in the nodes.

Chapter 3

The BYKa Scheme

3.1 Introduction

The Blom's scheme has many valuable features which are useful for low resourced sensor devices. However it has severe limitations which makes it impractical for large scale use. This chapter describes the original scheme pointing out its strengths and weaknesses and the modification to the scheme to overcome these limitations by using multiple keys in permutations with operations over a small finite field.

3.2 The Blom's Key Agreement Scheme

The Blom's scheme (Blom, 1983) (Blom, 1984) is a symmetric key scheme in which the nodes store an optimal amount of keying information so that it is unconditionally secure in the information-theoretic sense. An adversary with unlimited computing resources cannot break the scheme if the amount of captured keying material, corresponding to the number of compromised nodes, does not exceed a certain amount. There is simply insufficient information, and the success of breaking it is as good as any random chance. The attractiveness of the scheme lies in the simplicity of the computations and the

ability to authenticate each other.

Setup The Trusted Authority is responsible for all the keys. It generates its own secret master key $\mathbf{M}$ which is a random $(m \times m)$ symmetric matrix over a finite field $\mathbb{F}_p$. It assigns to each node one unique public key which is a $(m \times 1)$ column vector. For example, for nodes A and B, the public keys are $\mathbf{V}_A$ and $\mathbf{V}_B$ respectively. Using each node's public key together with the Trusted Authority's master key, the private keys for the nodes are generated as follows:

$$\begin{aligned} \text{Node } A\text{: } \mathbf{K}_A &= \mathbf{V}_A^T \cdot \mathbf{M} \pmod p \\ \text{Node } B\text{: } \mathbf{K}_B &= \mathbf{V}_B^T \cdot \mathbf{M} \pmod p \end{aligned}$$

Prior to deployment, the TA transfers the private keys to the nodes using a secure channel such as a direct cable connection. This procedure also allows the TA to physically identify and authenticate the node.

Pairwise Key Derivation After deployment, if a pair of nodes need to compute their pairwise key K_{AB}, they first obtain each other's public keys. These can be transmitted in plain text over the insecure channel. Using their counterpart's public keys with their own private keys, they compute a common pairwise key, K_{AB}. The process is shown as follows.

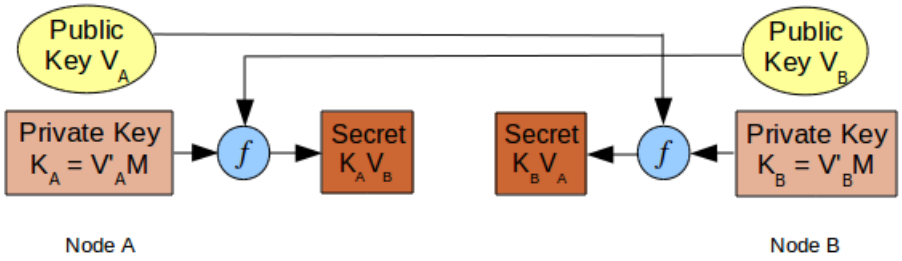


Figure 3.1: Blom's Key Agreement Scheme

Node A | Node B

$\xrightarrow{\mathbf{v}_A}$

$K_{BA} = \mathbf{K}_B \cdot \mathbf{V}_A \pmod p$
$= (\mathbf{V}_B^T \cdot \mathbf{M}) \cdot \mathbf{V}_A \pmod p$

$\xleftarrow{\mathbf{v}_B}$

$K_{AB} = \mathbf{K}_A \cdot \mathbf{V}_B \pmod p$
$= (\mathbf{V}_A^T \cdot \mathbf{M}) \cdot \mathbf{V}_B \pmod p$

Blom's Key Agreement Scheme

Correctness Node B computes the quantity $(\mathbf{V}_B^T \cdot \mathbf{M}) \cdot \mathbf{V}_A \pmod p$. This is a (1×1) scalar, and as the master key matrix $\mathbf{M}$ is symmetric, after transposing K_{BA} in node B, the two keys are identical.

$$\begin{aligned} \text{Node B:} \quad K_{BA}^T &= (\mathbf{V}_B^T \cdot \mathbf{M} \cdot \mathbf{V}_A)^T = \mathbf{V}_A^T \cdot \mathbf{M}^T \cdot \mathbf{V}_B \\ &= K_{AB} \end{aligned} \tag{3.1}$$

3.2.1 Features of the Blom's Scheme

The security of the Blom's scheme is analysed in detail in §4.3. Here, the attractive features for application in sensor networks are highlighted.

Implicit Mutual Authentication

The common identical pairwise key K_{AB} requires that both nodes have obtained their private keys from the TA. While any node, for example a rogue node E can send its public key $\mathbf{V}_E$ to node A, from which node A can compute a key K_{AE}, node E cannot compute K_{EA} without the private key $\mathbf{K}_E = \mathbf{V}_E^T \cdot \mathbf{M}$, since $\mathbf{M}$ is unknown. Hence, if node A can successfully decrypt a message encrypted with K_{AB}, it can trust that the node claiming to possess the public key $\mathbf{V}_B$ also possesses the private key, which could have only been generated by the TA. Of course, the key may have been stolen but this is a separate issue which is considered later. In effect, all the private keys share a common heritage which is the master key of the Trusted Authority. The nodes' private keys inherit their "genetic" material as a result of the computations using their public keys with the master key $\mathbf{M}$. There is implicit authentication as both parties contribute their keying material to compute the pairwise key. This is unlike the DH scheme where all the keying material is self generated in the nodes, or in the RSA scheme where only one party's public key and private key is used.

Immunity to the MITM attack

The Blom's scheme is a non-interactive scheme in the same sense as in (Paar & Pelzl, 2010) p. 398, wherein the two parties only need to exchange their public keys which are static and are public information. As there is implicit authentication, the process completes without further exchange. An adversary has nothing to manipulate except to send fictitious keys. The man-in-the-middle (MITM) attacker cannot succeed.

In the MITM attack, an adversary node E interposes itself between two nodes A and B. It poses as A to B, and similarly, as B to A. If this is successful, it acts as an intermediary between A and B, reading and modifying messages before forwarding them. In the Blom's scheme, if the attacker E forwards its own public key $\mathbf{V}_E$ to node A

impersonating node B, node A would compute the pairwise key K_{AE}. Node E cannot compute K_{EA} as it does not have the private key for $\mathbf{V}_E$. If E forwards $\mathbf{V}_B$ to node A, and $\mathbf{V}_A$ to node B, both nodes A and B can compute their pairwise key K_{AB}, but this cannot be obtained by node E. Messages encrypted between nodes A and B cannot be read by E. In this way, the Blom's scheme is immune to MITM attacks as both nodes must use keying material from the TA to compute their pairwise key.

Simple Computation

The pairwise key computation algorithm uses m modulo multiplications and $(m-1)$ additions and does not require large memory resources for RAM or the program code.

Unconditional Security

The system is said to be $(m-1)$ secure, i.e., a coalition of $(m-1)$ or less nodes pooling their keying material together cannot derive the pairwise key of any other pair of nodes (A. Menezes, van Oorschot, & Vanstone, 1996). By keeping the number of nodes deployed to be $< m$, the system is said to be *unconditionally secure* in the information theoretic sense since there is insufficient information to break the system. The scheme has a *capture threshold* m.

3.2.2 Weaknesses of the Blom's Scheme

The size of the pairwise secret key is the same as that of the elements of the master key matrix, i.e. data size used which is $log_2(p)$ bits. It would be necessary to make p sufficiently large to be of any practical use. The current NIST recommendations for use with AES is at least 112 bits and the Blom's scheme would need to implement large integer operations. Sensor devices are capable of operations using 8,16, 32, and even 64 bits.

To implement the scheme for large networks, the capture threshold m must be sufficiently large. Consequently the private key storage requirement in each node increases proportionally. The catastrophic failure that results if m or more nodes are captured caused Blom to remark, "it would be nice to have systems that degrade more gracefully but more research is needed" (Blom, 1983).

3.3 The BYka Scheme

Our scheme, called the Blom-Yang key agreement (BYka) scheme, uses the Blom's scheme (Blom, 1984) as the cryptographic primitive. It is modified by using multiple master keys in the Trusted Authority. Each node is also assigned multiple public keys , and the operations are over a small prime field $\mathbb{F}_p$, for example $p = 31$. All these keys are used together in permutations to compute multiple private keys for the node. After exchanging all their public keys, pairs of nodes compute their pairwise key using all the permutations of the node's private keys with the counterpart node's public keys. Both nodes would obtain a set of identical integers, not in the same order, which are used to form the pairwise key of sizes up to hundreds of bits. The BYka scheme is described as follows.

3.3.1 Base Station and Setup

The Trusted Authority is responsible for generating all the keys used in the network using the keying parameters; the number of master keys N and size m, the number of public keys η, the prime modulus for public key operations q, and the prime modulus for all other key operations p. For example using $N = 7,\ m = 16,\ \eta = 6,\ p = 31$ and $q = 65521$, it is possible to obtain pairwise keys of 128 bits for use in a network of about 10,000 nodes. All the nodes have to obtain their keys from the Trusted Authority. In this way, nodes have to first "touch base" with the Trusted Authority which also

ensures that the nodes are authentic before deployment.

Master Keys

The TA first generates its own N secret master keys $\mathbf{M}_1, \mathbf{M}_2, \cdots, \mathbf{M}_N$, each one being a random $(m \times m)$ symmetric matrix $\mathbf{M}$ over the prime field $\mathbb{F}_p$, where p is a small prime number, typically $p = 13, 17, 19, 23$ or 31. It is assumed that the random number generator is cryptographically secure and the random numbers are uniformly distributed over $[0, p-1]$.

Public Key-set and Public Key *ID*

The TA assigns to each node η unique public keys called the *public key-set*. Each public key-set consists of η $(m \times 1)$ column vectors of the Vandermonde matrix over the field $\mathbb{F}_q$. To cater for a large number of nodes, q must be much larger than p, for example $q = 65521$. As the elements of a column in the Vandermonde matrix are s^{i-1} for $i = 1, \cdots, m$, where s is called the "seed", the node's public key-set can be represented by η seeds $\{s, \cdots, s+\eta-1\}$. The seeds are consecutive and the smallest seed s is a multiple of η. In this way, all the public key-sets are unique and no two nodes share a common seed. The node's public key-set can then be succinctly represented by the smallest seed s which also serves as its public key *ID*, e.g. using $\eta = 6$, a node A with public key $ID_A = 240$ has public key seeds $\{240, 241, \cdots, 245\}$. Given a node's public key *ID*, anyone knowing q can generate its public key-set as follows,

$$\begin{aligned} \mathbf{V}_i^T &= \begin{bmatrix} 1 & s_i & s_i^2 \cdots s_i^{m-1} \end{bmatrix} \pmod{q} \\ \text{where} \quad s_i &= ID + i - 1, \quad \text{for } i = 1, \cdots, \eta \end{aligned} \tag{3.2}$$

When pairs of nodes exchange their public keys, they only need to transmit their *ID*s consisting of a few bits, e.g. 16 bits. This is an important feature saving time and

energy for the radio transmissions.

Private Key-set

The TA computes the "private keys" for each node using all the permutations of their η public keys with the N master keys to obtain the node's "private key-set" $\mathbf{S} = \{\mathbf{K}_{11}, \cdots, \mathbf{K}_{\eta N}\}$, where the private key $\mathbf{K}_{ij}$ is computed as follows,

$$\begin{aligned} \mathbf{K}_{_{ij}} &= \mathbf{V}_i^T \mathbf{M}_j \pmod p \\ \text{for } i &= 1, \cdots, \eta \text{ and } j = 1, \cdots, N \end{aligned} \tag{3.3}$$

PPMka The private key $\mathbf{K}_{ij}$ is computed from the i^{th} public key $\mathbf{V}_i$ and the j^{th} master key $\mathbf{M}_j$. We call the relationship of a private key with the public key and the master key used to compute it, the "private-public-master-key association" (PPMka).

Definition 4 (PPMka) *The private-public-master-key association (PPMka) specifies the particular public key that was used with a particular master key to compute the private key.*

Prior to deployment, the TA transfers the private key-set to the node using a secure connection and stores them in random order. Alternatively, the private key-set can be first shuffled before being transferred to the node. If a node is compromised and the private keys obtained, the adversary cannot tell from the storage location which public keys and master keys were used to compute them.

Choice of Public Key Seeds

Key Aliasing The number of public key seeds must be large enough to accommodate the network size. To do this, the public key operations are over a large field $\mathbb{F}_q$, for example, $q = 65521$ catering for about 10,000 nodes, but it can be much larger. As

the private key operations are over a small field $\mathbb{F}_p$, it is possible for multiple public keys to map to the same private key, a phenomenon we call "key aliasing" described as follows. Consider the private key $\mathbf{K}_k = \mathbf{V}_{s_n}^T \mathbf{M}_y$ where s_n is the seed for the public key $\mathbf{V}_n$. Denoting the elements of $\mathbf{M}_y$ as $M_{y_{ij}}$, and using Eqn. (3.2), the x^{th} element of the row vector $\mathbf{K}_k$ is,

$$\begin{aligned} \mathbf{K}_{k_x} &= \sum_{i=1}^{m} s_n^{i-1} M_{y_{ix}} \pmod{p} \\ &= M_{y_{1x}} + s_n^1 M_{y_{2x}} + \cdots + s_n^{m-1} M_{y_{mx}} \end{aligned} \tag{3.4}$$

For two nodes, say A and B, if any of their public key seeds are congruent, i.e. $s_A \equiv s_B \pmod{p}$, and for all $i = 0, \cdots, m-1$, the elements s_A^{i-1} and s_B^{i-1} are smaller than q, (the elements in the public key vectors do not "wrap round" q), then we have $s_A^{i-1} \equiv s_B^{i-1} \pmod{p}$ for all i. As a result, their private keys associated with the same master key are identical since,

$$\mathbf{K}_{A_x} = M_{y_{1x}} + s_A^1 M_{y_{2x}} + \cdots + s_A^{m-1} M_{y_{mx}} \pmod{p} \tag{3.5a}$$

$$\mathbf{K}_{B_x} = M_{y_{1x}} + s_B^1 M_{y_{2x}} + \cdots + s_B^{m-1} M_{y_{mx}} \pmod{p} = \mathbf{K}_{A_x} \tag{3.5b}$$

Master Key Leakage We also note in Eqn. (3.5), if a seed, $s_A \equiv 0 \pmod{p}$ and $r_A \equiv s_A^{i-1} \equiv 0 \pmod{p}$ for all i, then $K_{A_u} = M_{y_{1u}}$, i.e. the first row of the master key $\mathbf{M}_y$ leaks out in the u^{th} element of the private key $\mathbf{K}_A$. This can lead to other vulnerabilities.

Proposition 1 *A seed s_n is chosen such that,*

$$\left.\begin{array}{rl} \text{for some } w \leqslant m, & s_n^{w-1} > q \\ \text{i.e. } s_n^{w-1} & \equiv r_n \pmod{q} \\ \text{and } r_n & \not\equiv \left\{\begin{array}{l} 0 \pmod{p}, \text{ or} \\ s_n \pmod{p} \end{array}\right. \end{array}\right\} \tag{3.6}$$

3.3.2 Bootstrapping

The Trusted Authority installs into each node the "keying material" comprising the global keying parameters $\{m, N, \eta, p, q\}$, and the node's individual public key *ID*, and the private key-set. All these are static and can be stored in the ROM or flash memory. As noted before, the private keys are stored in a random order.

To assist with bootstrapping, the Trusted Authority can distribute subsets of the keying material to subsidiary key servers to help bootstrap the nodes. This requirement for nodes to first "touch-base" with the Trusted Authority or the subsidiary key servers to obtain their keying material ensures that nodes are authentic and can be trusted if they possess the keying material obtained from authentic sources.

If security demands that one compromised TA cannot comprise the whole system, a group or "committee" of TAs can be used. Each TA generates one subset of the N master keys that will be used to compute a subset of the node's private key-set. In this way, all the TAs must jointly endorse the node by contributing towards its private key-set. There is added security as one compromised TA will not break the entire system.

3.3.3 Pairwise Key Derivation

After deployment, any pair of nodes can compute their pairwise key after obtaining each other's *ID*s. For example, nodes A and B, having obtained each other's *ID*s, generate

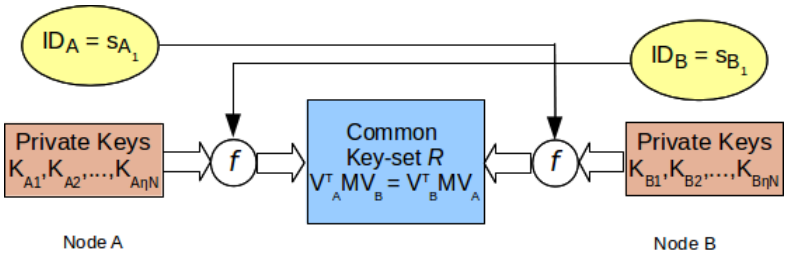


Figure 3.2: The BYka Process

their counterpart's public keys using Eqn. (3.2). Then, using all the permutations with its own private key-set, each node computes (mod p) the set R, called the "pairwise key-set", as follows,

$$\left.\begin{aligned}
&\text{For} \quad i,k = 1,\cdots,\eta, \quad \text{and} \quad j = 1,\cdots,N \\
&\text{Node A:} \quad s_{B_k} = ID_B + k - 1 \\
&\qquad \mathbf{V}_{B_k}^T = \begin{bmatrix} 1 & s_{B_k} & \cdots & s_{B_k}^{m-1} \end{bmatrix} \\
&\qquad R_{A_{ijk}} = \left\{\mathbf{K}_{A_{ij}}\mathbf{V}_{B_k}\right\} = \left\{(\mathbf{V}_{A_i}^T\mathbf{M}_j)\mathbf{V}_{B_k}\right\} \pmod p \\
&\text{Node B:} \quad s_{A_k} = ID_A + k - 1 \\
&\qquad \mathbf{V}_{A_k}^T = \begin{bmatrix} 1 & s_{A_k} & \cdots & s_{A_k}^{m-1} \end{bmatrix} \\
&\qquad R_{B_{ijk}} = \left\{\mathbf{K}_{B_{ij}}\mathbf{V}_{A_k}\right\} = \left\{(\mathbf{V}_{B_i}^T\mathbf{M}_j)\mathbf{V}_{A_k}\right\} \pmod p
\end{aligned}\right\} \tag{3.7}$$

The process is illustrated in Fig. (3.2).

Correctness of the MKB Scheme

Transposing the elements in Node B's pairwise key-set R_B gives,

$$R_{B_{ijk}} = (\mathbf{V}_{B_i}^T\mathbf{M}_j\mathbf{V}_{A_k})^T = \mathbf{V}_{A_k}^T\mathbf{M}_j^T\mathbf{V}_{B_i} \pmod p$$

The elements in R are (1×1) matrices, i.e. scalars, and the matrices $\mathbf{M}$ are symmetric i.e. $\mathbf{M}_j = \mathbf{M}_j^T$. Since the i, j, k are merely counters, then the numbers in R_A

and R_B are identical, though not arranged in the same order.

Constructing the Pairwise Key

The $N\eta^2$ numbers in the key-sets R are used by each node to obtain a common pairwise key. This is a long term pairwise key. Obtaining an identical key from the key-set may be done in several ways:

1. multiplying them together, or
2. sort the key-set values to obtain an ordered set, or
3. count the number of occurrences of the integers.

Multiplication: In this method all the elements are incremented by one to make them all non-zero $\in [1, p]$, and are then multiplied together modulo S_k, where S_k is a prime number of the desired key size, e.g. 128 bits, i.e.,

$$K_{pair} = \prod_{i=1}^{N\eta^2} R_{A_i} + 1 \text{ (mod } S_k) \tag{3.8}$$

Sorting: The key-set elements can be sorted in ascending or descending order. Both nodes would then obtain an identical key set. This can then be used to obtain a pairwise key by several means. A simple method is to just take a sufficient number of bits from the front or back of the list to form the pairwise key. Another method would use the sorted key-set as input to a hash function to obtain the pairwise key K_{pair}.

Occurrences: In this method, the number of occurrences of the integers are counted and used as input to a hash function. For example, if $p = 5$, $N = 3$ and $\eta = 2$, and the pairwise-key set contains $\{2, 1, 3, 2, 3, 2, 0, 0, 4, 3, 1, 2\}$, the occurrences of the respective integers $\{0, 1, 2, 3, 4\}$ gives the string "22431" which is used as input to a hash function to obtain the pairwise key K_{pair}.

Session Pairwise Key:

After the nodes have obtained their pairwise key K_{pair}, they use it to encrypt and exchange a randomly generated session key. After this, the long term pairwise key K_{pair} can be deleted for added security. The encryption algorithm uses recommended methods such as AES with HMAC (Barker et al., 2012). The MICAz mote is equipped with the CC2420 radio chip which includes a stand-alone AES encryption engine ("CC2420 Data Sheet", 2014).

3.4 Features of the BYka Scheme

3.4.1 Implicit Mutual Authentication

Overall, the BYka scheme is a mechanism that enables a pair of nodes to compute a set of identical numbers using their private keys with their counterpart's public key-set *ID*. It is an identity-based (more accurately, index based) symmetric key scheme and mutual authentication is a result of both nodes having obtained their keying material from the same TA.

MITM Attacks

In the man-in-the-middle (MITM) attack, an adversary E interposes between two nodes A and B, and entices A to believe it is B, and vice versa. It acts as an intermediary reading and modifying messages before forwarding them. This can succeed only if the messages are not protected using the endpoint keys K_{AB}, especially during the initial public key exchange phase.

Key agreement schemes in which a node generates its own public and private keys, and uses both of them to derive or protect the pairwise key are vulnerable. These include schemes such as the RSA and DH schemes. To mitigate this, additional measures to

authenticate the public keys must be used, such as using a trusted third party to sign the nodes' public keys.

The Blom's scheme is immune to MITM attacks. This is because the public and private keys are not generated by a node but by the TA. In addition, a node would use its own private key with the public key of its counterpart to derive their pairwise key. Both nodes must contribute part of the keying material in the pairwise key computation. In this way, the scheme is mutually authenticating. An adversary in the middle cannot form a pairwise key with either node as it does not have the private key associated with its (fabricated) public key. The attacker can only help to forward messages but not read them. The MITM attacker can only be an MITM helper.

3.4.2 Low Communication Overhead

The initial exchange of the public keys to begin the process is only the size of one public key-set *ID*, i.e. 16 bits for $q = 65521$, not considering the other overheads required such as destination *ID*, MAC addresses, etc. This makes the exchange fast and saves energy required for the radio transmissions.

3.4.3 Compact Code

The pseudo code is given in Listing (1). As can be seen it is very simple and can be coded using a few lines.

3.4.4 Memory Requirements

The key computation code is small due to its simplicity requiring a small amount of program memory ROM. The pairwise key is $N\eta mb$ bits in size. In our scheme, b is less than 8 bits. Using one byte for each element, requires $N\eta m$ bytes. Typically larger values of $N,\ \eta = 8$ and $m = 24$ require at most $1,536$ bytes for the private keys.

Listing 1: BYka Pairwise Key Computation Pseudo Code

Input: Neighbour Node's public key-set *ID*
Output: The pairwise BYka key K_{pair}
Initialise $K_{pair} = 1$;
Generate all the neighbour's public key-seeds;
for *each public key seed* **do**
  generate public key vector, $\mathbf{V}_i$ (mod q)
  for *each private key,* $\boldsymbol{K}_j$ **do**
    $R = \mathbf{K}_j \cdot \mathbf{V}_i$ (mod p) + 1 ;
    $K_{pair} = K_{pair} * R$ (mod S_k);
  end
end

3.4.5 Scalability

The limit on the network size, i.e. the number of nodes can be affected by the following factors: the private key keyspace, the public key keyspace, and the pairwise key keyspace. The Blom's capture threshold does not apply as will be shown in the next chapter.

Private Key Keyspace A private key has p^m possible values. A single node has $N\eta$ private keys. If the keys are all unique, there are $\frac{p^m}{N\eta}$ sets. If $p = 31,\ N = 8, \eta = 8$ and using a small value of $m = 12$, there are 1.23×10^{16} possible sets.

Public Key Keyspace The public key is a Vandermonde mode column vector with elements $s_n^{i-1}\ \ i = 1, \cdots, m$ satisfying Eqn. (3.6) to avoid private key aliasing. There are slightly less than q seeds after omitting some unsuitable seeds such as 0 and 1. Each node has η seeds, and the total number of public key-sets is then less than $\approx \frac{q}{\eta}$. If $q = 65521,\ \eta = 6$, it is possible to have approximately $10,900$ nodes. With 17-bit prime, $q = 131071$ we can have about 21000 nodes using $\eta = 6$. Using a 32-bit prime, e.g. $q = 4,294,967,291$ and $\eta = 6$, there are enough public key-sets for about 300×10^9 nodes.

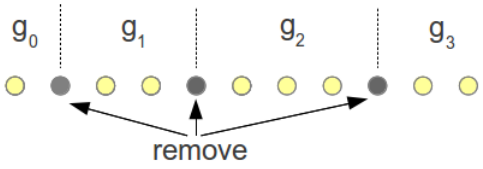


Figure 3.3: Partitioning 8 Items into 4 Groups

3.4.6 Pairwise Keyspace Size

In the BYka scheme, the pairwise key is formed from the pairwise key-set R consisting of $N\eta^2$ integers, each one $\in [0, p-1]$. The pairwise key can be up to $log_2(p^{N\eta^2})$ bits, well in excess of 128 bits. However, the effective key length is only as large as the number of possible combinations of the integers $\in [0, p-1]$ in the pairwise key-set. This effectively limits the number of the pairwise keys. The keyspace size can be obtained by counting all the possible combinations of the integers $0, 1, \cdots, p-1$, such that the total number of integers in each combination is exactly $N\eta^2$. This can be obtained by considering the following partitioning problem.

Partitioning Problem Given a row of $N\eta^2$ items, we wish to partition them into p groups $g_0, g_1, \cdots, g_{p-1}$ such that they contain the integers $0, 1, \cdots, p-1$ respectively. The number of items in each group represents the number of the respective integers. This is illustrated in Fig. (3.3) for the case of partitioning 8 items into 4 groups. To create the partitions, we first insert $(p-1)$ items into the row so that there are now $(N\eta^2 + p - 1)$ items. If any $(p-1)$ items are now removed, $(p-1)$ gaps would be created separating the remaining items into p groups as desired. The number of ways to remove $(p-1)$ items from $(N\eta^2 + p - 1)$ gives the keyspace size as follows,

$$K_{sp} = \binom{N\eta^2 + p - 1}{p - 1} \tag{3.9}$$

η	N	Values of p				
		13	17	19	23	31
4	3	40	48	52	59	71
	4	44	54	59	67	81
	5	48	59	64	73	89
	6	51	63	68	78	96
	7	53	66	72	82	102
	8	56	69	75	86	107
5	4	51	64	69	79	97
	5	55	68	74	86	106
	6	58	72	79	91	113
	7	61	76	82	95	119
	8	63	79	86	99	124

η	N	Values of p				
		13	17	19	23	31
6	4	57	71	78	90	111
	5	61	76	83	96	120
	6	64	80	88	102	127
	7	67	84	92	106	134
	8	69	87	95	111	139
7	6	69	87	95	111	140
	7	72	91	99	116	146
	8	74	94	103	120	152
8	6	74	93	102	119	151
	7	77	97	106	124	157
	8	79	100	109	128	163

Legend: Key Sizes 64 bits, 80 bits, 96 bits, 112 bits, 128 bits

Table 3.1: Keyspace in bits

The keyspace size in terms of bits is,

$$K_{sp} = log_2 \left[\frac{(N\eta^2 + p - 1)!}{(p-1)!(N\eta^2)!} \right] \tag{3.10}$$

Table (3.1) shows the keyspace in bits for various parameters. For example, with $p = 31$, and $N, \eta \geq 6$, the number of combinations in the keyspace exceeds 128 bits. The value of $p = 31$ is a good choice as it is a Mersenne prime which has some advantages for modulo operations.

In the original Blom's scheme, $K_{sp} = \binom{1+p-1}{p-1} = p$, i.e. the keyspace in bits is the same as the number of bits in p.

Example: A row of 8 items is to be placed into 4 groups g_1, g_2, g_3, g_4, each corresponding to integers $1, 2, 3, 4$. . There are $\binom{8+4-1}{4-1} = 165$ permutations. Fig.(3.3)shows one permutation comprising one of 1, two of 2's, three of 3's and two of 4's. If these are multiplied together to form the pairwise key, the key will be $(1^2 \times 2^2 \times 3^3 \times 4^2) = 1728$.

3.5 Summary

In the BYka scheme, unlike the original Blom's scheme, the TA and the nodes have multiple keys which are used in permutations to obtain multiple private keys which are stored in a random order in the node. The computations are over a small prime field. The pairwise key is also computed using all permutations of the counterpart's public keys with the private keys to obtain a large number of integers from which to obtain a large pairwise key. The scheme is mutually authenticating as both nodes must have obtained their keying material from the same TA to be able to compute their common pairwise key.

The node's public keys are a set of column vectors of the Vandermonde matrix selected such that the seeds are consecutive. This allows the set of public key seeds to be represented by the smallest seed, called the public key *ID*. This is only 16 to 32 bits long and sending it is fast, saving on energy for radio transmission. The memory, computation power, and energy requirements are well within the capabilities of today's sensor devices.

The important features of Blom's scheme including mutual authentication, immunity to the MITM attack, and simple computations are retained in the BYka scheme. In addition, computations use small data sizes of 8 bits and are able to obtain pairwise keys 128 bits and larger.

The number of nodes that can be deployed is limited by the size of the prime modulus q for the public key seeds. This can be large enough, for example 16 or 17 bits, to accommodate tens of thousands of nodes. The Blom's scheme is unconditionally secure only if there are less than m nodes deployed in the networks. In the BYka scheme, each node carries $N\eta$ private keys bringing the capture threshold down to just $\frac{m}{\eta}$ nodes. However, each private key needs to be correctly associated with the master key and public key used to compute it, i.e. the PPMka information must be found. The

next chapter shows how, without this PPMka information, the capture threshold does not apply, allowing the BYka scheme to break free of the capture threshold of the original Blom's scheme.

Chapter 4

Security Analysis

4.1 Introduction

The original Blom's scheme is unconditionally secure if the number of nodes is less than the capture threshold. The BYka scheme appears to have a lower threshold since each node carries several keys. However, even if a very powerful adversary is able to capture and extract the keys from a sufficient number of nodes, it will not be able to break the scheme. This is due to the ambiguities introduced by the numerous private keys, stored in a random storage, and computed over a small field. These make the private keys unusable directly, thereby preventing the adversary from mounting attacks as in the original Blom's scheme. The security of the BYka scheme is broken down and analysed in these three areas:

(1). The vulnerabilities of the keys – master keys, private keys, and the pairwise keys;
(2). The vulnerabilities of the Blom's scheme as applicable in the BYka scheme;
(3). The resilience against system breakdown due to node capture.

This chapter analyses the first two vulnerabilities, and the resilience analysis is left for the next chapter.

4.2 Vulnerability of Keys

It is important that an adversary, having obtained the keying parameters cannot fabricate any of the keys. For this part, we consider that the adversary can only monitor and read all encrypted messages. It knows the keying parameters N, η, p and q, but cannot steal the private keys. The adversary can resort to discovering the keys using brute force where all the possible keys are fabricated and, by trial-and-error, test each one until eventually finding the correct one. To defeat this, the keys must be sufficient large and randomly distributed so that an infeasible amount of resources will be required to try all the possible keys. As a benchmark, the NIST (Barker et al., 2012), p. 27, recommends that the number of operations that is required to break a cryptographic algorithm or system should be in excess of 2^{128} or 3.4×10^{38} steps for equivalent security strength of 128 bits.

4.2.1 Resistance Against Brute Force Attacks

The Master keys

It is assumed that the TA has access to a good random number generator for generating the master key matrices. Hence, the elements of the master keys can be assumed to be a uniform distribution of random integers $\in [0, p-1]$. Each master key has $\frac{m(m+1)}{2}$ unique elements and each element is $\in [0, p-1]$ giving the keyspace of $N_k = p^{\frac{m(m+1)}{2}}$. For typical values of $p = 31,\ m = 12,\ N = 6$, there are 2.119×10^{116} possible master keys. To mount an attack by brute force, the attacker has to choose N master keys from this keyspace, which assuming there are no repetitions, has about $\binom{N_k}{N}$ possible sets. Using the above values, there are 1.26×10^{695} or 2^{2309} sets of master keys, clearly an infeasible effort using the brute force attack.

4.2.2 Public Keys

The public keys are $(m \times 1)$ column vectors of the Vandermonde matrix as given in Eqn. (3.6). They are public and there is no need to protect them in any way. The public keys are exchanged by sending the public key *ID*, a single seed value of 16 or more bits. This is in plain text. An adversary can learn the public keys of the transmitting node. Apart from this, no other useful information is obtained.

The adversary may assume any arbitrary *ID* and entice legitimate nodes to compute a pairwise key with it. Nothing is gained by the adversary but to cause the legitimate node to expend some resources. This is the DoS attack. Other mechanisms not covered in this study are required to mitigate this attack.

4.2.3 Private Keys

The private key is computed from $\mathbf{K} = \mathbf{V}^T\mathbf{M}$ where $\mathbf{V}^T = [\ 1 \quad s \quad \cdots \quad s^{m-1}\ ]$. The elements of $\mathbf{M}$, generated using a cryptographically secure random number generator, are uniformly distributed over $\mathbb{F}_p$.

As shown in §3.3.1, the seeds s_n are chosen such that

$$\left.\begin{array}{rl} \text{for some } w \leqslant m, & s_n^{w-1} > q \\ \text{i.e. } s_n^{w-1} & \equiv r_n \pmod q \\ \text{and } r_n & \not\equiv \left\{\begin{array}{l} 0 \pmod p, \text{ or} \\ s_n \pmod p \end{array}\right. \end{array}\right\} \tag{3.6}$$

Randomness of the Private Keys The x^{th} element of the private key $\mathbf{K}_{k_x}$, associated with the seed s_n and master key $\mathbf{M}_y$, from Eqn. (3.4) can be written as,

$$\begin{aligned} K_{k_x} &= \sum_{i=1}^{m} s_n^{i-1} \pmod q \, M_{y_{ix}} \pmod p \\ &= M_{y_{1x}} + s_n M_{y_{2x}} + s_n^2 M_{y_{3x}} + \cdots + r_n M_{y_{mx}} \pmod p \end{aligned} \tag{4.1}$$

The terms making up the x^{th} element of $\mathbf{K}_k$ in Eqn.(4.1) are not all zeros if the values of s are chosen in compliance with Eqn. (3.6). As they are the sums of products of integers with elements of the master key which are uniformly distributed random integers, the operations being over the prime field $\mathbb{F}_p$, they are also random integers. Hence, all the elements of the private key $\mathbf{K}_k$ are also random integers $\in [0, p-1]$.

Brute force attack In the brute force attack, the adversary would attempt to construct the private key-set and use it to masquerade a node. Due to the randomness of the private keys, there are p^m possible private keys from which the attacker would choose $N\eta$ keys to form the private key-set. There are $\binom{p^m}{N\eta}$ possible combinations, assuming each one is unique. Even with small values of $m = 12$, $p = 13$ and $N, \eta = 4$, there are $\frac{(p^m)!}{(p^m - N\eta)!(N\eta)!} = 1.22{\times}10^{147}$ possibilities. Clearly, the adversary has a very small chance of fabricating a valid private key-set to use in a rogue node, or to masquerade as a legitimate node.

4.2.4 Pairwise Keys

Key Lengths The pairwise key is derived from a set of $N\eta^2$ numbers $\in [0, p-1]$, called the "pairwise key-set", R. For nodes A and B with seeds s_A and s_B respectively that comply with Eqn. (3.6), an element in the key-set $R_{A_{ijk}}$ is,

$$\begin{aligned} R_{A_{ijk}} &= (\mathbf{V}_{A_i}^T \mathbf{M}_j)\mathbf{V}_{B_k} = \mathbf{K}_{A_{ij}}\mathbf{V}_k \text{ (mod } p) \\ &= \sum_{u=1}^{m} K_{A_{ij_u}} s_{B_k}^{u-1} \\ \text{for } i,k &= 1, \cdots, \eta \text{ and } j = 1, \cdots, N \end{aligned}$$

The elements of the private keys $K_{A_{ij}}$ are shown to be random integers in §4.2.3. Similarly, the elements in the key-set R_A are also random integers being the sums and products of random integers in $\mathbb{F}_p$.

From §3.4.6, the pairwise keyspace can be sufficiently large in excess of 128 bits. Each one can be up to $log_2(p^{N\eta^2})$ bits long. Thus, brute force attacks on the pairwise keys are not feasible.

4.3 Security of the Blom's Scheme

The adversary can attempt to break the scheme by capturing enough nodes and extracting their keys. It is assumed that the adversary has very powerful computing resources and is able to physically take control of the nodes to extract the keying material from ROM and RAM. However the adversary cannot steal the master keys. This section analyses how the Blom's scheme may be broken so that countermeasures can be found.

If the adversary is able to obtain the private keys from a sufficient number of nodes, he can attempt to break the scheme in two ways; use the captured keys to construct completely new valid keys for use in rogue nodes, or to derive the master keys and hence completely break the system. The identity theft attack where a node is cloned using captured keys is not in the scope of this study.

4.3.1 Masquerade Attacks – Sybil Attacks

In the masquerade or Sybil attack, the adversary fabricates new valid public and private keys and uses them for crafting new nodes to masquerade as legitimate nodes in the network. From the previous section §4.2.1, these keys cannot be feasibly fabricated by trial and error. However, if enough nodes are compromised and their keys obtained, the adversary can try to use these keys to construct new valid keys.

Consider that n nodes and their public and private keys,$\{\mathbf{V}_1, \mathbf{K}_1\}, \cdots, \{\mathbf{V}_n, \mathbf{K}_n\}$

have been obtained. The attacker can fabricate a new public key $\mathbf{V}_X$ by linear combination of the captured keys using suitable coefficients $\alpha_1, \cdots, \alpha_n$, i.e.,

$$\mathbf{V}_X = \alpha_1 \mathbf{V}_1 + \cdots + \alpha_n \mathbf{V}_n \pmod{p} \tag{4.2}$$

The corresponding private key $\mathbf{K}_X$ would be a similar linear combination of the captured private keys,

$$\begin{aligned} \mathbf{K}_X &= \mathbf{V}_X^T \mathbf{M} = (\alpha_1 \mathbf{V}_1^T + \cdots + \alpha_n \mathbf{V}_n^T)\mathbf{M} \\ &= \alpha_1 \mathbf{V}_1^T \mathbf{M} + \cdots + \alpha_n \mathbf{V}_n^T \mathbf{M} \\ &= \alpha_1 \mathbf{K}_1 + \cdots + \alpha_n \mathbf{K}_n \pmod{p} \end{aligned}$$

The attacker will be able to fabricate any public key and the corresponding private key for use in a rogue node X. This node can obtain a valid pairwise key with a legitimate node. For example, node X and node A exchanged their public keys. Node X computes the pairwise key,

$$\begin{aligned} K_{XA} &= \mathbf{K}_X \mathbf{V}_A = [\alpha_1 \mathbf{K}_1 + \cdots + \alpha_n \mathbf{K}_n]\mathbf{V}_A \\ &= \alpha_1 \mathbf{K}_1 \mathbf{V}_A + \cdots + \alpha_n \mathbf{K}_n \mathbf{V}_A \\ &= \alpha_1 \mathbf{V}_1^T \mathbf{M} \mathbf{V}_A + \cdots + \alpha_n \mathbf{V}_n^T \mathbf{M} \mathbf{V}_A \pmod{p} \end{aligned}$$

Similarly, node A computes,

$$\begin{aligned} K_{AX} &= \mathbf{K}_A\mathbf{V}_X = \mathbf{K}_A[\alpha_1\mathbf{V}_1 + \cdots + \alpha_n\mathbf{V}_n] \\ &= \alpha_1\mathbf{K}_A\mathbf{V}_1 + \cdots + \alpha_n\mathbf{K}_A\mathbf{V}_n \\ &= \alpha_1\mathbf{V}_A^T\mathbf{M}\mathbf{V}_1 + \cdots + \alpha_n\mathbf{V}_A^T\mathbf{M}\mathbf{V}_n \pmod p \\ \text{After transposing,} \quad K_{AX}^T &= \alpha_1\mathbf{V}_1^T\mathbf{M}^T\mathbf{V}_A + \cdots + \alpha_n\mathbf{V}_n^T\mathbf{M}^T\mathbf{V}_A \pmod p \\ &= K_{X1} \quad \text{since } \mathbf{M} = \mathbf{M}^T \end{aligned}$$

4.3.2 Requirements of Public Keys:

To defend against this attack, the public keys must meet these three conditions:

1. the public keys have a prescribed format,
2. the public keys must be linearly independent of each other,
3. the number of captured nodes n must be less than m.

Public Key Format

The first condition ensures that a key formed from arbitrary linear combinations of captured keys would not be accepted. For example, without any prescribed format, a public key may be $\mathbf{V}_1 = \begin{bmatrix} 2 & 4 & 1 \end{bmatrix}$. It can be linearly combined with $\mathbf{V}_2 = \begin{bmatrix} 5 & 6 & 2 \end{bmatrix}$, to obtain a new public key $\mathbf{V}_x = 2\mathbf{V}_1 + 3\mathbf{V}_2 = \begin{bmatrix} 19 & 26 & 8 \end{bmatrix}$. If $\mathbf{V}_x$ is acceptable as a public key, a new node with corresponding private key $\mathbf{K}_X = 2\mathbf{K}_1 + 3\mathbf{K}_3$ can be introduced into the network.

On the other hand, if all the public keys are columns of the Vandermonde matrix, arbitrary public keys of other formats would simply be invalid and are discarded

In spite of this feature, the adversary may still be able to construct public keys $\mathbf{V}_X$ whose elements are of the correct format by solving for α by combining the captured

keys as follows,

$$\mathbf{V}_X = \alpha_1\mathbf{V}_1 + \cdots + \alpha_n\mathbf{V}_n \pmod{p} \tag{4.3a}$$

Writing the elements in $\mathbf{V}_A^T$ as $\begin{bmatrix} V_{A_1} & \cdots & V_{A_m} \end{bmatrix}$,

$$\text{i.e.} \quad \begin{bmatrix} V_{X_1} \\ V_{X_2} \\ \vdots \\ V_{X_m} \end{bmatrix} = \begin{bmatrix} V_{1_1} & \cdots & V_{n_1} \\ V_{1_2} & \cdots & V_{n_2} \\ \vdots & \ddots & \vdots \\ V_{1_m} & \cdots & V_{n_m} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} \pmod{p} \tag{4.3b}$$

$$\mathbf{V}_X = \begin{bmatrix} \mathbf{V}_1 & \mathbf{V}_2 & \cdots & \mathbf{V}_n \end{bmatrix} \vec{\alpha} \pmod{p}$$

$$\text{i.e.} \quad \mathbf{V}_X = \mathbf{V}\vec{\alpha} \pmod{p}$$

where

$$\mathbf{V} = \begin{bmatrix} V_{1_1} & \cdots & V_{n_1} \\ V_{1_2} & \cdots & V_{n_2} \\ \vdots & \ddots & \vdots \\ V_{1_m} & \cdots & V_{n_m} \end{bmatrix} \quad \text{and} \quad \vec{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$$

Linearly Independent Public Key Vectors

If all the public key vectors $\mathbf{V}_1, \cdots, \mathbf{V}_n$ are linearly independent of each other, then if $n < m$, by definition $\mathbf{V}_X$, cannot be a linear combination of other vectors. The solution to Eqn. (4.3b) is trivial, i.e. $\alpha_1, \cdots, \alpha_n = 0$ if the number of captured keys n is less than the master key matrix size m, and the Sybil attack cannot succeed.

However, if the number of captured nodes $n \geq m$, then using any m of the captured

nodes, Eqn. (4.3b) can be written as,

$$\vec{\alpha} \;=\; \mathbf{V}^{-1}\mathbf{V}_X \pmod{p} \tag{4.4}$$

Since $\mathbf{V}$ is a square $(m \times m)$ matrix with linearly independent columns, for example, the Vandermonde matrix, then $\mathbf{V}^{-1}$ exists, the determinant $|\mathbf{V}| \neq 0$, and the solution for $\vec{\alpha}$ is determinate and non-trivial.

Hence, if the public key vectors are not linearly independent, the Sybil attack can succeed. In addition, if the public key vectors are linearly independent, the Sybil attack can only succeed if m or more keys are obtained. Therefore, to prevent the Sybil attack, the public key vectors must be linearly independent and no more than m nodes can be captured.

4.3.3 Attacking the Master Key

The attacker can also use the capture keys to compute the master key itself. There are two possible approaches.

Brute Force Using a Single Captured Node

It is known that a private key is computed as $\mathbf{K} = \mathbf{V}^T\mathbf{M} \pmod{p}$. The attacker having obtained the keys from a captured node, can construct an arbitrary master key and use it to compute a trial private key using the node's public key. If this matches the node's private key, then the master key is found. If not, the process is repeated. This will succeed eventually after trying all the possibilities. However, the number of possible master keys, $p^{\frac{1}{2}m(m+1)}$ is prohibitively large using suitable keying parameters of m and p. For example, even when using small values of $p = 13$ and $m = 16$, the number of trials possible is 7.72×10^{86}. This is infeasible with current computing resources.

Using a Sufficient Number of Captured Nodes

Consider that m nodes have been captured and all the public keys are linearly independent. Let the elements of the master key **M** be M_{ij} as follows:

$$\mathbf{M} = \begin{bmatrix} M_{11} & \cdots & M_{1m} \\ \vdots & \ddots & \vdots \\ M_{m1} & \cdots & M_{mm} \end{bmatrix}$$

Consider a node n with public key $\mathbf{V}_n$ and private key $\mathbf{K}_n = \mathbf{V}_n^T\mathbf{M}$. Since **M** is symmetric, after transposing, this can be written as,

$$\mathbf{M}\mathbf{V}_n = \mathbf{K}_n^T$$

All the private keys from the m captured nodes can be combined and written as,

$$\mathbf{M}\begin{bmatrix} \mathbf{V}_1 & \mathbf{V}_2 & \cdots & \mathbf{V}_m \end{bmatrix} = \begin{bmatrix} \mathbf{K}_1^T & \mathbf{K}_2^T & \cdots & \mathbf{K}_m^T \end{bmatrix}$$

$$\text{i.e. } \mathbf{M}\mathbf{V} = \mathbf{K}$$

Where

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_1 & \mathbf{V}_2 & \cdots & \mathbf{V}_m \end{bmatrix} = \begin{bmatrix} \mathbf{V}_{11} & \mathbf{V}_{21} & \cdots & \mathbf{V}_{m_1} \\ \mathbf{V}_{12} & \mathbf{V}_{22} & \cdots & \mathbf{V}_{m_2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{V}_{1m} & \mathbf{V}_{2m} & \cdots & \mathbf{V}_{m_m} \end{bmatrix}$$

and

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1^T & \mathbf{K}_2^T & \cdots & \mathbf{K}_m^T \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{21} & \cdots & \mathbf{K}_{m_1} \\ \mathbf{K}_{12} & \mathbf{K}_{22} & \cdots & \mathbf{K}_{m_2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{K}_{1m} & \mathbf{K}_{2m} & \cdots & \mathbf{K}_{m_m} \end{bmatrix}$$

If the matrix $\mathbf{V}$ is invertible, the master key $\mathbf{M}$ can be obtained,

$$\mathbf{M} = \mathbf{K}\mathbf{V}^{-1} \tag{4.5}$$

If the number of nodes captured is m, and all the public key vectors are linearly independent, for example, being columns of the Vandermonde matrix, then the $(m \times m)$ matrix $\mathbf{V}$ with linearly independent columns is non singular with non-zero determinant and the inverse $\mathbf{V}^{-1}$ exists. The master can be then derived and the scheme completely broken.

A simpler approach without computing the inverse matrix is to construct the system of linear equations and solve for $\mathbf{M}$. The private key for node n is given as,

$$\mathbf{K}_n = \mathbf{V}_n^T \begin{bmatrix} M_{11} & \cdots & M_{1m} \\ \vdots & \ddots & \vdots \\ M_{m1} & \cdots & M_{mm} \end{bmatrix} = \begin{bmatrix} K_{n1} & \cdots & K_{nm} \end{bmatrix}$$

Using all the m captured nodes, the system of equations can be assembled,

$$\begin{bmatrix} \mathbf{V}_1^T & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{V}_1^T \\ \mathbf{V}_2^T & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{V}_m^T \end{bmatrix} \begin{bmatrix} M_{11} \\ \vdots \\ M_{1m} \\ M_{22} \\ \vdots \\ M_{mm} \end{bmatrix} = \begin{bmatrix} K_{11} \\ \vdots \\ K_{1m} \\ K_{21} \\ \vdots \\ K_{mm} \end{bmatrix} \tag{4.6}$$

and solved using, for example, the Gaussian elimination method.

4.3.4 Effort Required to Break the Scheme

The adversary has to do several tasks to break the scheme, see Fig. (4.1). Consider that the public and private keys of the captured nodes are already available. First, the keys must be assembled into the $(m \times m)$ system of equations. Then, these are solved to obtain a possible solution for the master key. This must be tested to see if it is correct by using it with one of the public keys to compute the private keys. This is compared against the captured keys. If correct, then one of the master keys is found. If not, another public key is used. The whole process is repeated until all the master keys are found.

Gaussian Elimination

The Gaussian elimination method can be used to solve the system of equations. The number of operations to solve an $(m \times m)$ system of linear equations involves $(\frac{m^3}{3} + m^2 + \frac{m}{3})$ multiplications and $(\frac{m^3}{3} + \frac{m^2}{2} - \frac{5m}{3})$ additions (Tapia, Lanius, Mac Zeal, & Parks, 2001). Simplifying this by considering only the multiplications, there are about $\frac{m^3}{3}$ operations. With $m = 16$ it requires approximately $1,365$, say 10^3 operations.

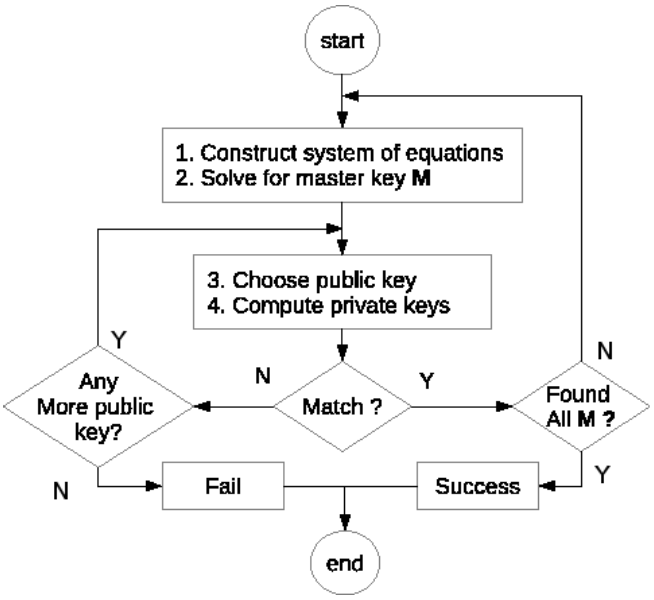


Figure 4.1: Steps Required to Obtain the Correct Master Keys

4.3.5 Limitations of the Blom's Scheme

As the devices are not provided with tamper proof mechanisms, any number of nodes can be captured by a determined and powerful adversary and have their keys stolen. While using linearly independent public keys prevents the adversary from mounting the Sybil attack, this cannot be prevented if m or more nodes are compromised. The master key itself can be derived.

The Bloms's scheme is said to be $(m-1)$ secure in that if the number of nodes deployed is $< m$, even if the entire network is compromised, the master key cannot be obtained. It is then unconditionally secure in the information theoretic sense with a "capture threshold" of m. Its practicality is limited, since a large m requires proportionally large memory in each node to store the private key which is a row vector with m elements.

4.4 Security of the BYka Scheme

4.4.1 Capture Threshold

The BYka scheme uses the Blom's scheme as the cryptographic primitive and would appear to inherit its weakness – the capture threshold. In fact the capture threshold is lower as each node has $N\eta$ private keys. In order to capture enough nodes to construct the N $(m \times m)$ systems of linear equations to solve for the master keys or mount the Sybil attack, the adversary needs to capture only $\lceil \frac{m}{\eta} \rceil$ nodes. However, this apparent lower capture threshold is only effective if each captured private key can be correctly associated with the public key and the master key used to compute it.

4.4.2 Private-Public-Master-Key Association (PPMka)

In the Blom's scheme using only a single master key and public key, the association of the single private key to the public key and master key is obvious. However, in the BYka scheme, each node has $N\eta$ private keys, each computed from one of the TA's N master keys and one of the node's η public keys, and each one is stored in random order in the node. For each captured node, the adversary is able to obtain the keying parameters N, η, m, p, q, the $N\eta$ private keys, the public key *ID* and hence the η seeds forming the public keys. Before any of the private keys can be used, the related public key and master key must be known.

4.4.3 Indiscernibility of the Private-Public-Master-Key Association

In §4.2.3, it was shown that the elements of the private keys are random integers $\in [0, p-1]$. For example, let the seed s_n satisfying Eqn. (3.6) be used as the public key seed for node n. The x^{th} element of the private key $\mathbf{K}_k$ computed with the master key

$\mathbf{M}_y$ can be written as,

$$\begin{aligned} K_{k_x} &= \sum_{u=1}^{m} s_n^{u-1} \pmod{q}\, M_{y_{ux}} \pmod{p} \\ &= M_{y_{1x}} + s_n M_{y_{2x}} + s_n^2 M_{y_{3x}} + \cdots + r_n M_{y_{mx}} \pmod{p} \end{aligned}$$

Each element is a random integer $\in [0, p-1]$, being the sum of products of s_n^{u-1} and r_n with the uniformly distributed random integers $M_{y_{iu}}$, the operations being over the prime field $\mathbb{F}_p$. The private key K_k is a row vector of random integers $\in [0, p-1]$, i.e.

$$\mathbf{K}_k = \begin{bmatrix} K_{k_1} & \cdots & K_{k_u} & \cdots & K_{k_m} \end{bmatrix}$$

As a result, the adversary, by examining $\mathbf{K}_k$, would not find any information about the public key seed used to compute it. Each of the $N\eta$ private keys is indistinguishable from another.

Destruction of the PPMka

The TA computes and stores the node's $N\eta$ private keys in a random order, extinguishing any possibility of determining the PPMka from the storage order. The PPMka is unknown, indiscernible, and irretrievable by examining the keys or its storage location. The relationships between the private keys and the associated public keys and master keys are ambiguous.

Without the PPMka information, each key can only be associated with the correct master key and public key with a probability of $\frac{1}{N\eta}$, no better than a random chance. To solve for one of the master keys, the correct PPMka for m private keys must be found. The overall probability $\frac{1}{(N\eta)^m}$, can be made extremely small using suitable parameters. The indiscernibility of the PPMka is an important side-effect that will be exploited to make the BYka scheme resilient against node capture attacks.

Security Implications Due to the Unknown PPMka

The original Blom's scheme is unconditionally secure if two conditions are satisfied: the number of captured private keys is less than m, and the public keys are linearly independent. In the light of the BYka scheme, a third condition may be stated: each captured private key must be correctly associated with the master key and public key used to create it. This is of course trivial in the Blom's scheme with a single key.

The unknown PPMka enables the BYka scheme to break free from the capture threshold and remain virtually unconditionally secure for network sizes in excess of the capture threshold. In addition, the probability of discovering the PPMka can be made so small that the BYka scheme is secure even if a large number of nodes can be captured.

4.4.4 Resilience Against Sybil Attacks

Consider how the adversary can mount the Sybil attack after capturing (enough) Nm private keys. It is also known that the public keys used are $\mathbf{V}_{C_1}, \cdots, \mathbf{V}_{C_m}$, and each one is a column vector of the Vandermonde matrix. The adversary would choose an arbitrary public key ID_X with seed s_{X_1} and construct the public key as $\mathbf{V}_{X_1}^T = \begin{bmatrix} 1 & s_{X_1} & \cdots & s_{X_1}^{m-1} \end{bmatrix}$ such that,

$$\begin{aligned} \mathbf{V}_{X_1} &= \alpha_{1_1}\mathbf{V}_{C_1} + \cdots + \alpha_{1_m}\mathbf{V}_{C_m} \pmod q \\ &= \begin{bmatrix} \mathbf{V}_{C_1} & \cdots & \mathbf{V}_{C_m} \end{bmatrix} \vec{\alpha} \end{aligned} \tag{4.7}$$

The above $(m \times m)$ system of equations can be solved to obtain $\vec{\alpha}$ since $[\ \mathbf{V}_{C_1} \quad \cdots \quad \mathbf{V}_{C_m}\ ]$ is a $(m \times m)$ Vandermonde matrix which has a non-zero determinant as all the columns are linearly independent. The private key associated with $\mathbf{M}_1$ and $\mathbf{V}_{X_1}$ can then be

constructed as linear combinations of the captured private keys as follows,

$$\begin{aligned}\mathbf{K}_{X_{1_{M1}}} &= \mathbf{V}_{X_1}^T \mathbf{M}_1 \\ &= \left(\alpha_{1_1}\mathbf{V}_{C_1}^T + \cdots + \alpha_{1_m}\mathbf{V}_{C_m}^T\right)\mathbf{M}_1 \\ &= \alpha_{1_1}\mathbf{K}_{C_{1_{M1}}} + \cdots + \alpha_{1_m}\mathbf{K}_{C_{m_{M1}}} \end{aligned} \tag{4.8}$$

Here, the private key $\mathbf{K}_{C_{1_{M1}}}$ is associated with the public key $\mathbf{V}_{C_1}$; $\mathbf{K}_{C_{2_{M1}}}$ with $\mathbf{V}_{C_2}$; and so on, and all are associated with the same master key, $\mathbf{M}_1$. As shown earlier each private key is just a row vector of random integers and there is no information about their PPMka. From the $\frac{m}{\eta}$ captured nodes, the number of possible ways of associating each key with the public keys and the master key $\mathbf{M}_1$ is Φ_1 as follows.

$$\Phi_1 = \left[\frac{(N\eta)!}{(N\eta-\eta)!}\right]^{\frac{m}{\eta}}$$

The other private keys associated with master keys $\mathbf{M}_2, \cdots, \mathbf{M}_N$ must be similarly computed. After the private keys associated with each master key are obtained, they are removed, leaving fewer keys for the next round. The total number of possible solutions to fabricate the private keys for $\mathbf{K}_{X_1}$ is,

$$\Phi = \sum_{i=0}^{N-1}\left[\frac{(N\eta-i\eta)!}{(N\eta-i\eta-\eta)!}\right]^{\frac{m}{\eta}} \tag{4.9}$$

The values of Φ are given in Table (4.1) for values of p, N and η which give pairwise key sizes of 64 bits or more.

To complete the attack, the private keys for the other public keys, $\mathbf{V}_{X_2}, \cdots, \mathbf{V}_{X_\eta}$ must also be done. The total number of possible sets of private keys is thus Φ^η. This makes the Sybil attack impossible as it is done interactively, one at a time.

η	N	Master key size m			
		12	16	24	32
6	6	2.16×10^{18}	2.84×10^{27}	3.90×10^{36}	7.61×10^{54}
	7	1.64×10^{19}	5.67×10^{28}	2.07×10^{38}	2.91×10^{57}
	8	9.45×10^{19}	7.46×10^{28}	6.30×10^{39}	4.79×10^{59}
7	6	1.97×10^{22}	2.55×10^{33}	3.43×10^{44}	4.65×10^{55}
	7	2.07×10^{23}	8.37×10^{34}	3.55×10^{46}	1.53×10^{58}
	8	1.57×10^{24}	1.68×10^{36}	1.90×10^{48}	2.20×10^{60}
8	6	2.41×10^{26}	2.41×10^{26}	3.55×10^{39}	5.37×10^{52}
	7	3.52×10^{27}	3.52×10^{27}	1.91×10^{41}	1.08×10^{55}
	8	3.54×10^{28}	3.54×10^{28}	5.88×10^{42}	1.03×10^{57}

Table 4.1: Number of Solutions Φ for Pairwise Key Sizes $\geq$ 64 bits

4.4.5 Resilience Against Attacks on the Master Keys

Similarly, due to the unknown PPMka, the adversary would not be able to compute the master keys even if a sufficient number of private keys was available. Using the same consideration as in §4.4.4 above, for each master key say $\mathbf{M}_1$, the adversary needs to assemble $(m\times m)$ equations from the private keys, all correctly associated with $\mathbf{M}_1$ and the public keys. From each node there are $\frac{(N\eta)!}{(N\eta-\eta)!}$ ways to arrange the η private keys according to $\mathbf{M}_1$ and the η public keys. The total number of ways to obtain these equations from the $\frac{m}{\eta}$ nodes is Φ_1, similar to Eqn. (4.9). Since all the N master keys must be obtained and used together, the total number of possible master key solutions is also given as in Eqn. (4.9). The total number of possible solutions, Φ is also given in Table (4.1). Once all the master keys are found, the adversary can fabricate all the necessary keys as desired. By choosing suitable keying parameters, for example $m = 24, N = 7, \eta = 8$, the number of possible solutions is 6.3×10^{39}. If this is the most efficient attack, the security strength would be 132 bits.

4.5 Other Security Issues

4.5.1 Immunity to MITM Attacks

As shown in §3.2.1, the Blom's scheme is immune to the MITM attack because it is a non-interactive scheme and nodes need to only exchange their publicly known public key *ID*s, and both nodes use each other's public keys with their own private keys to compute the pairwise key. For this to succeed, the keying material must come from the same TA. The BYka scheme is similar in operation and inherits the Blom's scheme immunity to the MITM attacks.

4.5.2 Key Escrow

The TA is a key escrow entity. It holds the master keys which can be used to compute the pairwise keys of any pair of nodes. All messages between the nodes in the network can be decrypted by the TA. This may not be a desirable feature in terms of privacy. However in commercial organisations, this may be desirable since the management must be able read all messages within the organisation.

The key escrow attribute may be relinquished by the TA deleting all or some of the master keys after computing all the possible private keys that are anticipated to be used. However, this is a drastic step and as will be shown later, the master keys cannot be recovered.

4.5.3 DoS Attacks

The initial public key *ID* exchange messages are in clear text. An adversary can eavesdrop and learn the public keys of the nodes but there is no consequence. However, the adversary can attempt to mount DoS attacks. An attacker can attempt to deplete a node's resources by sending it fictitious public key *ID*s in the DoS attack. The adversary

would monitor messages, and if enough *ID*s are learnt, determine the number of public keys used, η. In addition the value of η can be learnt from captured nodes. It is then easy to fabricate a valid public key-tag *ID* which is an integer factor of η and send this to the target node to cause it to compute the pairwise key. This thesis does not address the mitigations at this stage other than to suggest that the identity of the rogue node can be disseminated to other nodes and blacklisted.

4.5.4 Compromised-key Impersonation Attacks

If a node is compromised, it can be exploited in two ways. First, the adversary can impersonate node C to any other node. This is the identity theft attack. Secondly, the adversary can mount the compromised-key impersonation attack. Here, if an adversary node E has obtained node C's keys, the adversary E can impersonate any node to interact with the compromised node C. This attack can happen as follows:

Assume that node E has the set of private keys $\mathbf{K}_C$ and public key ID_C belonging to node C. Node E wishes to obtain the pairwise key with node C, impersonating node G. They exchange their public key *ID*s:

$$\begin{aligned} E \rightarrow C \quad &: \quad ID_G \quad \text{(node } E \text{ claims to be node } G\text{)} \\ C \rightarrow E \quad &: \quad ID_C \end{aligned}$$

The nodes would normally generate the counterpart's public keys and compute their pairwise key. However, while node C uses the received ID_G to compute the pairwise key using its private key-set $\mathbf{K}_C$, unknown to node C, node E also uses the same public key ID_G with its stolen private key-set $\mathbf{K}_C$.

$$\begin{aligned} C: \text{ generates } \mathbf{V}_{G_{1,\cdots,\eta}}, \text{ computes } K_{CG} &= \mathbf{K}_{C_{1,\cdots,N\eta}} \mathbf{V}_{G_{1,\cdots,\eta}} \\ E: \text{ generates } \mathbf{V}_{G_{1,\cdots,\eta}}, \text{ computes } K_{GC} &= \mathbf{K}_{C_{1,\cdots,N\eta}} \mathbf{V}_{G_{1,\cdots,\eta}} \end{aligned}$$

The two pairwise keys are naturally identical and node C has no way of knowing of node E's deception. An additional mechanism to detect and retire compromised nodes needs to be incorporated. Further research would be required.

4.5.5 Forward Secrecy

A cryptographic system is said to have forward secrecy if previously recorded messages cannot be decrypted should the long term pairwise keys be compromised. For example, the long term pairwise key between nodes A and B is K_{AB}. It was used to exchange a session key K_{sAB} for all subsequent messages. If the key K_{AB} is obtained, and all previous messages were recorded, the adversary would be able to discover the session key K_{sAB} and decrypt all the messages between nodes A and B. The BYka scheme does not have forward secrecy.

4.5.6 Key Revocation

Stolen Keys

A node can be captured and its keying material used to create a rogue node and redeployed into the network. This study does not include how to mitigate such impersonation or identity theft attacks.

However, assuming that a detection system is in place, the detected *ID* can be disseminated throughout the network and embargoed. The affected node can either be discarded or retrieved and provided with new keys.

4.6 Summary

The security of the BYka scheme is analysed in terms of the strengths of the keys used, the vulnerabilities of the underlying Blom's scheme, and how it may be attacked by an

adversary who is able to capture any number of nodes and obtain their keying material.

The master keys, private keys, and pairwise keys are random and large, making the brute force attacks to fabricate these keys infeasible. The number of possible pairwise keys can be made sufficiently large by selecting suitable keying parameters for N, η, m and p.

The underlying Blom's scheme is unconditionally secure if there are less than m nodes in the network. If the public key vectors are linearly independent, the Sybil attack cannot succeed. However, if more than m nodes are deployed, assuming that they can be captured and their keying material obtained, capturing m nodes will enable the adversary to mount the Sybil attack as well as to derive the master key. This is because each node has only one private key and it is obviously computed using the single public and master key. The captured private keys can be used to construct one system of $(m \times m)$ linear equations which can be solved to obtain the master key.

On the other hand, in the BYka scheme, each node has multiple private keys and each one is indistinguishable from each other. Each private key has a small probability of being correctly associated with the public key and master key used to compute it. By using a suitable number of master and public keys, the probability of correctly associating each of the $N\eta$ private keys to each of the η public keys and N master keys can be made so small that there is an infeasibly large number of possibilities.

If the PPMka cannot be feasibly obtained, the BYka scheme would be virtually unconditionally secure. No matter how many nodes are compromised, the Sybil and master key attacks cannot commence. This opens up another possibility of securing the BYka scheme. Since the adversary cannot use the captured keys directly to mount the Sybil attack, and linearly independent public keys were required as a countermeasure, it may be possible to relax this requirement so that public keys are not necessarily linearly independent. This will also make the master key attack even more difficult as the adversary will then need to capture even more nodes to find those with linearly

independent vectors in order to solve for the master keys. More research using this idea may make the scheme even more secure and computationally simpler.

The BYka inherits the immunity of Blom's scheme against MITM attacks as well as its weaknesses including the lack of forward secrecy and the vulnerability to the compromised-key impersonation attack. Other mechanisms must be found to mitigate these vulnerabilities.

The unknown private-public-master-key-associations (PPMka) becomes the linchpin for security in the BYka scheme to break free from the bounds of the original Blom's scheme. The next chapter examines whether and how the PPMka can be discovered.

Chapter 5

Cryptanalysis of the PPMka

5.1 Introduction

The PPMka indiscernibility is the linchpin to make the BYka scheme secure. It makes the captured private keys useless for the Sybil and master key attacks since these require that the public keys and master keys used to compute them are known. In addition, the PPMka obscurity can be engineered to a desirable security level. The probability of finding the correct PPMka can be made so small that it requires a huge amount of effort even if a very large number of nodes can be captured and their keys extracted.

The PPMka cannot be found from examining the information in the nodes. What is left to do is for the adversary to use pairs of nodes to compute their pairwise key, and by observing the internal results of the computations, obtain clues about the PPMka. We first show the scenario where master keys can be obtained successfully. Then we show how, by selecting suitable parameters, the circumstances can be engineered to make this extremely difficult. We obtained analytical results to estimate the effort required, both in the number of compromised nodes required, and the number of possible master key solutions using the most efficient attack.

5.2 Pairwise Key-set Attack

The attacks to solve for the master keys using brute force with only information captured from nodes, as shown in §4.4.5, involve an infeasibly large number of iterations if suitable keying parameters are used. A better approach is to study the internal interactions between pairs of nodes as they compute their pairwise keys. As the integers forming the pairwise keys are identical across the two nodes, these can be identified and used to infer the public keys and master keys associated with the private keys linked to these identical integers. This attack is called the "pairwise key-set attack".

Definition 5 (Pairwise key-set attack) *The adversary, given a pair of captured nodes, e.g. nodes A and B, uses each other's public keys to compute the key-sets R_A and R_B. Then, the matching integers in R_A and R_B are identified and can be used to link the related private keys to the public keys and master keys, revealing the PPMka.*

5.2.1 Without Ambiguities

If the pairwise key-set has only distinct integers, then the attack will successfully identify the PPMka of the private keys in the two nodes. The attack proceeds as follows. The attacker takes a pair of nodes, and using each other's public keys, computes the key-sets $\{R_A\}$ and $\{R_B\}$. This is illustrated in Fig. (5.1) for the simple case with $N = 2,\ \eta = 2$, assuming all the integers in the key-sets are distinct. Both sets will have identical integers but in a different order. By linking the identical integers across both sets, the private keys producing the matching integers are both associated with the same master key, and each private key must be associated with the public key used in the computation.

For example in Fig.(5.1), we find for pair 1,

$$\begin{aligned} \mathbf{K}_{A_1}\mathbf{V}_{B_1} &= \mathbf{K}_{B_1}\mathbf{V}_{A_1} \\ \text{i.e.} \quad (\mathbf{V}_{A_1}^T\mathbf{M}_x)\mathbf{V}_{B_1} &= (\mathbf{V}_{B_1}^T\mathbf{M}_x)\mathbf{V}_{A_1} \\ \therefore \mathbf{K}_{A_1} = \mathbf{V}_{A_1}^T\mathbf{M}_x \quad &\text{and} \quad \mathbf{K}_{B_1} = \mathbf{V}_{B_1}^T\mathbf{M}_x \end{aligned}$$

In the same way for all the other identical integer pairs, using all the private keys in nodes A and B, all the PPMka for the private keys in nodes A and B can be found. By successive pairing with other nodes, and if all the pairwise key-sets have distinct integers, the PPMKa of a sufficient number of private keys can be obtained and the master keys derived from the solution to a system of equations formed from these private keys. The pairwise key computations are over the field $\mathbb{F}_p$. If p is a large prime, for example $p = 65521$, then the probability that all the integers in the pairwise key-sets are distinct is very high, for example 70% with $N = 6, \eta = 6$.

5.2.2 With Ambiguities

If the numbers in the pairwise key-set R are not all unique, we say there are "collisions". This can arise because each element in $R \in \mathbb{F}_p$ can only take one of p values. Fig. (5.2) illustrates the case for $N = 2,\ \eta = 2$ where three numbers in R_A and R_B are identical. Collisions give rise to ambiguities for the PPMka. For example, in Fig. (5.2), multiple associations in nodes A and B are possible but all of them cannot be correct.

Definition 6 *A collision occurs when two or more integers in the pairwise key-set* $R_X = \{\boldsymbol{K}_{X_i} \cdot \boldsymbol{V}_{Y_j}\}$ *are identical.*

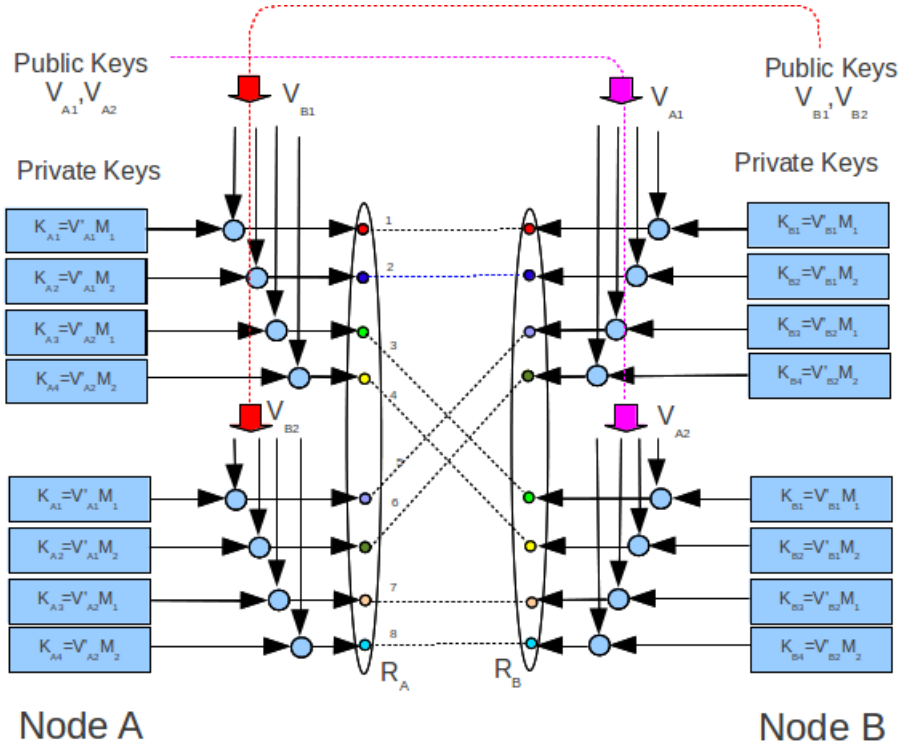


Figure 5.1: Key-set Attack Without Ambiguities, for Case $N = 2, \eta = 2$

Probability of Collisions

There are $N\eta^2$ elements in R, and each one is $\in [0, p-1]$. The probability that all the numbers in R are unique, i.e. no collision, is,

$$P_u = \frac{p}{p}\frac{p-1}{p}\cdots\frac{p-\overline{N\eta^2-1}}{p} = \frac{p!}{(p-N\eta^2)!}p^{-N\eta^2}$$

This probability can be made very small by choosing suitable keying parameters. For example using $p = 31$ with small values of $N = 3$ and $\eta = 3$, we have $P_u = 1.9537 \times 10^{-8}$. With typical values of $p = 31,\ N = 7,\ \eta = 6$, there are 252 numbers in R and, with only 31 numbers to use, numerous collisions are certain. The key-set attack using all the public keys to compute the key-set containing $N\eta^2$ elements $\in [0, p-1]$ can result in lots of collisions giving rise to ambiguities. A more efficient attack should be used.

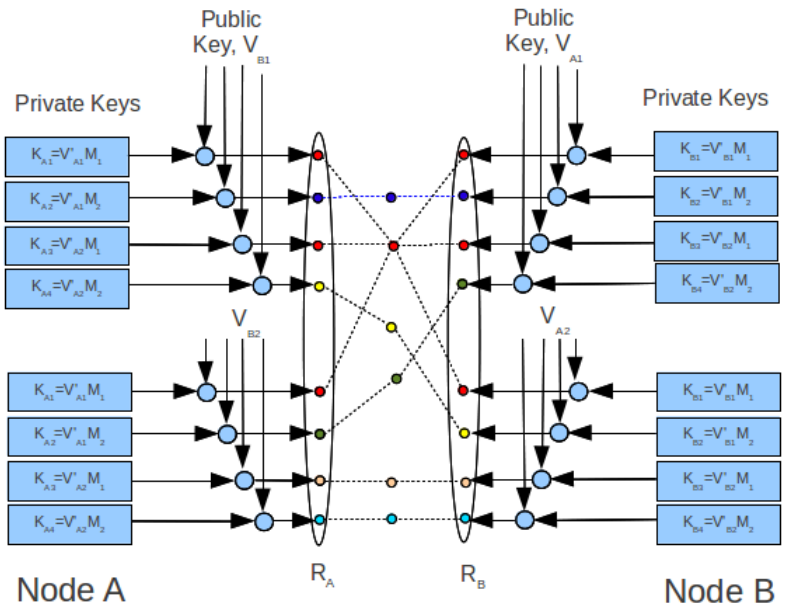


Figure 5.2: Key-set Attack with Collisions, for $N = 2, \eta = 2$

5.2.3 Pairing Attacks

To increase the chance of successfully identifying the PPMka using a pair of nodes, the number of integers that can collide should be reduced. This means the number of integers in the pairwise key-set should be made as small a possible. This can be done by using only one public key each time to compute a partial pairwise key-set R_r in each node. Now, the "partial key-sets" formed, R_{rA} and R_{rB}, contain only $N\eta$ elements reducing the probability of collisions.

A most Efficient Attack We call this the "pairing attack". It results in the smallest meaningful set of integers that can be used. For instance, if the integers are obtained by using one public key with one or a small selection of the private keys, there are very few integers, resulting in less chance of collision. However, there is no information on which of the private keys should be chosen since the PPMKa are ambiguous.

Fig. (5.3) illustrates the pairing attack, showing only one of the key-set numbers for clarity. Here, node A computes $\mathbf{K}_{A1}\mathbf{V}_{B2}$ which is identical to node B's $\mathbf{K}_{B3}\mathbf{V}_{A2}$. This

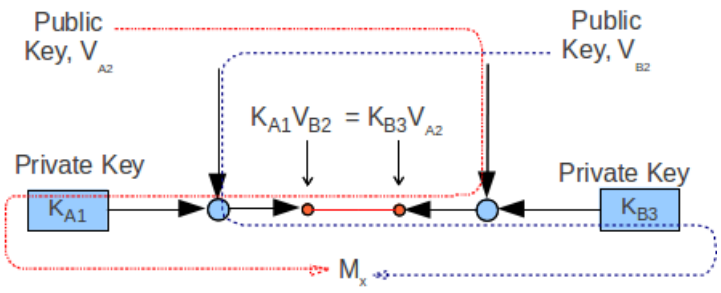


Figure 5.3: Pairing Attack Operation

creates a "circuit" linking $\mathbf{V}_{A2}, \mathbf{K}_{A1}, \mathbf{M}_x, \mathbf{K}_{B3}$ and $\mathbf{V}_{B2}$. The circuit reveals the PPMka for the private keys since ,

$$\begin{aligned} \mathbf{K}_{A1}\mathbf{V}_{B2} &= \mathbf{K}_{B3}\mathbf{V}_{A2} \\ \text{i.e. } (\mathbf{V}_{A2}^T\mathbf{M}_x)\mathbf{V}_{B2} &= (\mathbf{V}_{B2}^T\mathbf{M}_x)\mathbf{V}_{A2} \\ \text{then, } \mathbf{K}_{A1} = \mathbf{V}_{A2}^T\mathbf{M}_x \quad &\text{and} \quad \mathbf{K}_{B3} = \mathbf{V}_{B2}^T\mathbf{M}_x \end{aligned}$$

Couplers and Couplings

The partial key-sets formed from the pairing attack contain identical integers across both sets. This set of integers is called the set of couplers, see Fig. (5.4). In the ideal case there should be exactly N couplers across both sets, one for each of the master keys, for example using $\mathbf{V}_{B_1}$ in A and $\mathbf{V}_{A_1}$ in B,

$$\begin{aligned} \text{Node A:} \quad & \mathbf{V}_{A_1}^T\mathbf{M}_i\mathbf{V}_{B_1}, \cdots, \mathbf{V}_{A_1}^T\mathbf{M}_N\mathbf{V}_{B_1} \\ \text{Node B:} \quad & \mathbf{V}_{B_1}^T\mathbf{M}_i\mathbf{V}_{A_1}, \cdots, \mathbf{V}_{B_1}^T\mathbf{M}_N\mathbf{V}_{A_1} \end{aligned}$$

However, there may be more, due to the small field $\mathbb{F}_p$.

Definition 7 (Coupler) *A coupler is defined as an identical integer that occurs in both key-sets R_{rA} and R_{rB}. A set of couplers is the set of these integers.*

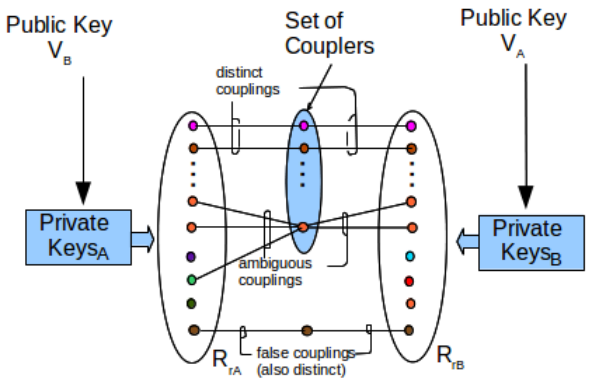


Figure 5.4: Couplers and Couplings

Definition 8 (Coupling) *A coupling is defined as the link connecting a coupler to the identical integers in both key-sets.*

A coupler has one or more couplings on each side.

Types of Couplers and Couplings: Due to the computations over a small prime field $\mathbb{F}_p$, it is possible for many couplers to occur, see Fig. (5.4). The following types of couplings can be observed:

1. Distinct Couplings There is only one coupling on each side of the coupler. This may, or may not, correctly link the private keys to the public keys. For the pairing attack to succeed, there must be exactly N couplers, each with distinct couplings.

2. Ambiguous Couplings There are multiple couplings on either side of the coupler. This results in many possible links connecting the private keys to the public key.

Attack Strategies

We see that by taking a pair of nodes and using each other's public key one at a time to compute the partial key-sets, the related private keys can be identified with the public

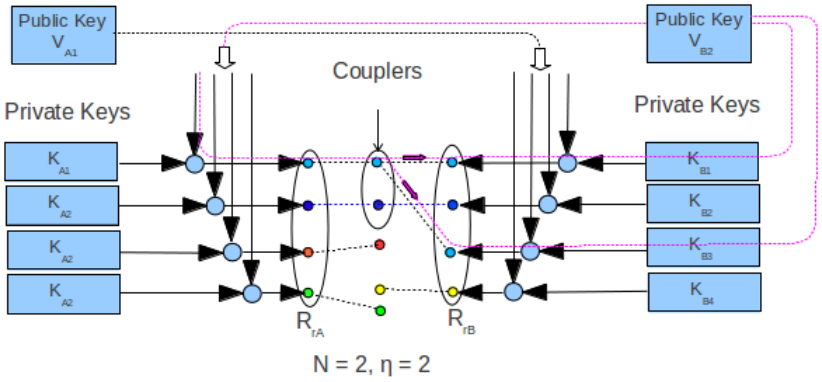


Figure 5.5: Pairing Attack with Collisions

keys used, if the partial key-sets yield exactly N couplers. Each private key is also clearly associated with one of the N master keys. This can be repeated using as many captured nodes as necessary to reveal all the PPMka. On the other hand, if there are too many collisions and the number of couplers is $> N$, even though some of them have distinct couplings, it is not possible to conclusively link the related private key to the public key used since it might be a false one.

Alternatively, each pairing produces a certain number of couplers, N_c, each one possibly correctly linking the private key to the public key used. Compared to the brute force case, the number of possibilities for the correct PPMka is now reduced since $N_c \leq N\eta$. By trying all the possible PPMka, the adversary will be able to find the correct one if the effort is feasible. For this case, the adversary needs to capture sufficient number of nodes, $\lceil \frac{m}{\eta} \rceil$.

We consider two different strategies covering both ends of the spectrum. First, the "unlimited capture" where the adversary captures as many nodes as required, and the second approach, the "limited capture" in which only a relatively few nodes but sufficient number of nodes are used.

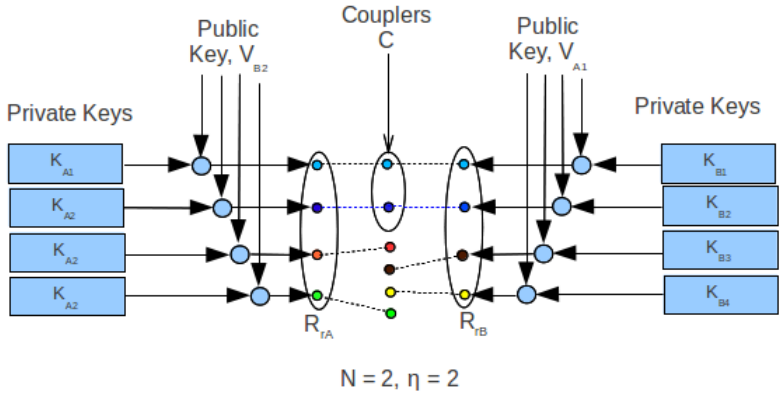


Figure 5.6: Pairing Attack without Collision

5.3 Pairing Attack with Unlimited Capture

If a pairing attack produces a partial key-set which has no ambiguity, the corresponding N private keys can be correctly associated with the public key used. Each private key is also associated with one of the master keys. By pairing the exposed node with other nodes using the previously found private keys, if no collisions occur, all the PPMka's will eventually be found. However, due to the small field $\mathbb{F}_p$ and a large number of elements $N\eta > p$ in the partial key-sets R_r, collisions are certain.

5.3.1 The Traitor Node

The pairing attack will be successful if each pairing results in non-colliding key-sets. However, with suitable choice of keying parameters, this probability is very small. The attack would have a better chance of success if one node can be found such that all the N private keys associated with one public key, say $\mathbf{V}_1$ is known. This set of private keys can be used to reduce the ambiguities in subsequent pairings. We call this node the "traitor node", since it can be used to betray other nodes. For example in Fig. (5.6), both nodes A and B are possible traitor nodes.

Definition 9 (Traitor Node) *A traitor node is one in which the PPMka of all N private*

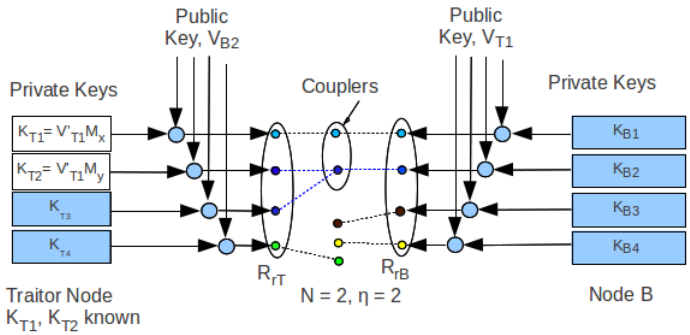


Figure 5.7: Traitor Node Can Be Used to Attack the PPMka

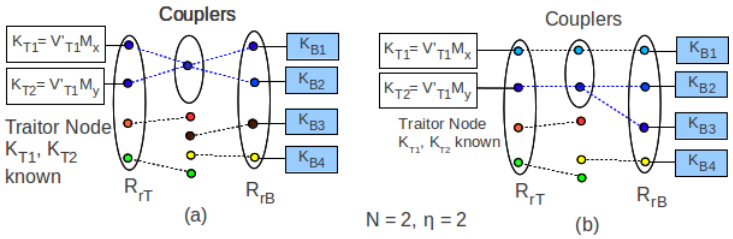


Figure 5.8: Traitor Node Cannot be Used to Attack the PPMka

keys associated with the N master keys, and all associated with one of the public keys, are known.

Use of the Traitor Node Using the traitor node, another node say B, is paired with it, see Fig. (5.7). If the number of couplings in R_{rB} is N, they distinctly link the related private keys in B to the exposed private keys in T revealing the PPMka, i.e. $\mathbf{K}_{B1}$ and $\mathbf{K}_{B2}$ must be associated with $\mathbf{M}_x$ and $\mathbf{M}_y$ respectively, and both associated with public key $\mathbf{V}_{B2}$.

This is not so straightforward if the number of couplings or couplers in R_{rB} is $\neq N$. The PPMka of the keys related to colliding couplers will be ambiguous, as in Fig. (5.8). Fig. (5.8a) shows the partial key-set R_{rB} having only 1 coupler. While $\mathbf{K}_{B1}$ and $\mathbf{K}_{B2}$ can both be associated with $\mathbf{V}_{B2}$, their associations with the master keys are ambiguous. In Fig. (5.8b), R_{rB} has more than N couplings, i.e. 3 instead of 2. Now it is not clear whether $\mathbf{K}_{B2}$ or $\mathbf{K}_{B3}$ is associated with $\mathbf{V}_{B2}$ and master key $\mathbf{M}_y$.

Proposition 2 (Existence of a traitor node) *The reduced key-sets of a pair of nodes*

A and B are obtained by multiplying one of each other's public keys with its own private keys, i.e.

$$\begin{aligned} R_{rA} &= \boldsymbol{K}_{A_{1,\cdots,N\eta}} \boldsymbol{V}_{B_i} \\ R_{rB} &= \boldsymbol{K}_{B_{1,\cdots,N\eta}} \boldsymbol{V}_{B_j} \end{aligned}$$

A traitor node is found if and only if, either one of reduced key-sets R_{rA} or R_{rB} has N couplings with the couplers. The node whose reduced key-set has exactly N couplings is the traitor node.

In Fig. (5.9a), node A has exactly N couplings so all the private keys are associated with the same public key, and each one is associated with one of the N master keys. Node A is a traitor node. However, node B has 4 private keys which can be associated with the 3 master keys. There are $\binom{4}{3} = 4$ possible PPMka and B is not a traitor node.

5.3.2 Finding a Traitor Node

To find a traitor node, a pair of nodes is taken, and using one of the counterpart's public keys, the partial key-sets R_{rA} and R_{rB} are computed, for example, see Fig. (5.9) for the case $N = 3$. There are $N = 3$ couplers (two are repeated). Set R_{rA} has $N_c = N = 3$ couplings, and R_{rB} has $N_c = 4$. Let R_C contain the couplers. The reduced key-sets R'_{rA} and R'_{rB} are formed by excluding the elements belonging to R_C. The node A whose reduced set is disjoint with R_C is a candidate as a traitor node.

In general, a traitor node can be found if;

1. Set R'_{rA} is disjoint with $(R'_{rB} \cup R_C)$, or
2. Set R'_{rB} is disjoint with $(R'_{rA} \cup R_C)$, or
3. Sets R'_{rA}, R'_{rB}, and R_C are all disjoint with each other.

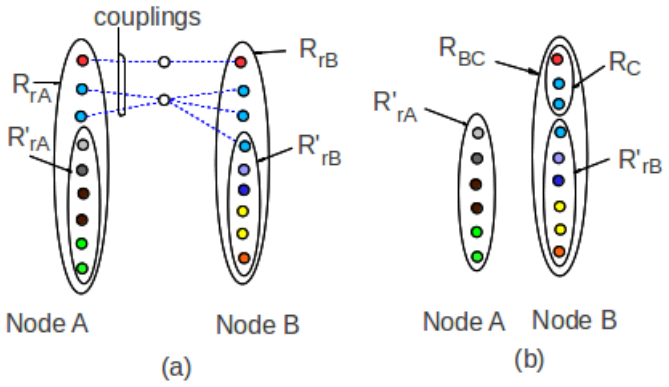


Figure 5.9: Finding the Traitor Node

The integers in the key-sets are random and uniformly distributed and, by counting the number θ_t of possible arrangements satisfying the above conditions, the probability of finding a traitor node can be computed as a fraction of the total number of all possible arrangements. The following counting problem enables the number of arrangements of traitor nodes to be obtained.

Equivalent Counting Problem

To obtain the probability of finding a traitor node in the pairing attack, the number of possible arrangements of the following equivalent combinatorial problem is considered.

Permutations of r Integers $Q_{N_a r}$

Before proceeding to count the occurrences of traitor nodes, the quantity, $Q_{N_{ar}}$ will be required. This quantity $Q_{N_{ar}}$ is the number of permutations of N_a integers taken from r integers such that, in each case, all the r integers are used without any being omitted. For example, in arranging 4 integers given the 3 integers $\{1, 2, 3\}$, permutations like $\{1, 1, 2, 3\}$ and $\{1, 2, 2, 3\}$ will be included, but excludes permutations using only one or two of the integers such as $\{1, 1, 1, 1\}$ and $\{1, 1, 2, 1\}$, etc. The number of permutations $Q_{N_a r}$ can be obtained by considering cases where $r = 1$, $r = 2$, $r = 3$, etc., integers are used in each case, as follows.

Using $r = 1$: Consider that there is only one integer used to arrange in N_a places. The number of permutations is $Q_{N_a 1} = 1^{N_a 1} = 1$.

Using $r = 2$: With 2 integers to use, there are 2^{N_a} permutations but these include $\binom{2}{1}[1^{N_a}]$ permutations which use only one integer. Hence, omitting those with only one integer,

$$Q_{N_a 2} = 2^{N_a} - \binom{2}{1}\left[1^{N_a}\right] = 2^{N_a} - \binom{2}{1}\left[Q_{N_a 1}\right]$$

Using $r = 3$: With 3 integers to use, the 3^{N_a} permutations include arrangements that have only 1 and 2 integers as well. Hence, permutations which have **all** 3 numbers are,

$$\begin{aligned} Q_{N_a 3} &= 3^{N_a} - \binom{3}{1}\left[1^{N_a}\right] - \binom{3}{2}\left[2^{N_a} - \binom{2}{1}[1^{N_a}]\right] \\ &= 3^{N_a} - \binom{3}{1}Q_{N_a 1} - \binom{3}{2}Q_{N_a 2} \\ &= 3^{N_a} - \sum_{i=1}^{2}\binom{3}{i}Q_{N_a i} \end{aligned}$$

General Case In general, the number of permutations of N_a integers using r integers, in which repeats are permitted, but **all** the r integers are used, is,

$$Q_{N_a r} = r^{N_a} - \sum_{i=1}^{r-1}\binom{r}{i}Q_{N_a i} \text{ where } Q_{N_a 1} = 1 \tag{5.1}$$

5.3.3 Traitor Node Permutations

In Fig. (5.9), let the number of integers in sets R'_{rA}, R'_{rB} be N_a and N_b respectively. The number of integers in set R_C is $N_c = N$, and $N_a = N_b = N\eta - N$. The traitor nodes may be found if the two sets R'_{rA} and R_{BC} are disjoint, or R_{AC} and R'_B are disjoint, or all three sets R'_{rA}, R'_{rB} and R_C are disjoint. These give the number of traitor node permutations.

Two Disjoint Sets

Consider the case where R_A is disjoint with R_{BC}. The set R'_{rA} has N_a, set $R_{BC} = (R'_{rB} \cup R_C)$ has $N\eta$ elements. Consider now the various possible cases.

(a). R'_{rA} has 1 Distinct Integer If set R'_{rA} uses only 1 distinct integer, with all 1s', 2s', etc., the number of permutations using p integers of R'_{rA} is $\binom{p}{1}1^{N_a}$. The remaining integers are used in R_{BC} in $(p-1)^{N\eta}$ possible ways. The number of arrangements possible is then,

$$\begin{aligned} \theta_{u1} &= \binom{p}{1} \times 1^{N_a}(p-1)^{N\eta} \\ &= \binom{p}{1} \times Q_{N_a 1} \times (p-1)^{N\eta} \\ \text{where} \quad Q_{N_a 1} &= 1^{N_a} \end{aligned}$$

(b). R'_{rA} has 2 Distinct Integers If R'_{rA} uses only 2 distinct integers taken from p integers, the number of permutations of R'_{rA} is $\binom{p}{2}\left[2^{N_a} - \binom{2}{1}Q_{N_a 1}\right]$, excluding those with only 1 integer such as $\{1, 1, \cdots, 1\}$, $\{2, 2, \cdots, 2\}$, etc. The number of permutations using 2 integers in R'_{rA} disjoint with R_{BC} is then,

$$\begin{aligned} \theta_{u2} &= \binom{p}{2}\left[2^{N_a} - \binom{2}{1}1^{N_a}\right](p-2)^{N\eta} \\ &= \binom{p}{2}\left[2^{N_a} - \binom{2}{1}Q_{N_a 1}\right](p-2)^{N\eta} \\ &= \binom{p}{2}Q_{N_a 2}(p-2)^{N\eta} \\ \text{where} \quad Q_{N_a 2} &= \left[2^{N_a} - \binom{2}{1}Q_{N_a 1}\right] \end{aligned}$$

(c). R'_{rA} has 3 Distinct Integers If R'_{rA} has 3 distinct integers, there are $\binom{p}{1}$ repeats of single integers, $\binom{p}{2}$ repeats of two integers which in turn has $\binom{2}{1}$ repeats of single integers which must be removed. Then, the number of possible arrangements is,

$$\begin{aligned}
\theta_{u3} &= \binom{p}{3}\left\{3^{N_a} - \left[\binom{3}{1}\times 1^{N_a}\right] - \left[\binom{3}{2}\times\left(2^{N_a} - \binom{2}{1}\times 1^{N_a}\right)\right]\right\}(p-3)^{N\eta} \\
&= \binom{p}{3}\left\{3^{N_a} - \left[\binom{3}{1}\times Q_{N_a1}\right] - \left[\binom{3}{2}\times Q_{N_a2}\right]\right\}(p-3)^{N\eta} \\
&= \binom{p}{3}Q_{N_a3}(p-3)^{N\eta} \\
&\quad \text{where} \quad Q_{N_a3} = 3^{N_a} - \sum_{i=1}^{3-1}\binom{3}{i}Q_{N_ai}
\end{aligned}$$

(d). R'_{rA} has r Distinct Integers In general, if R'_{rA} has r distinct integers, the total number of unique permutations excluding those with less than r distinct integers is,

$$\begin{aligned}
\theta_{ur} &= \binom{p}{r}\left\{r^{N_a} - \left[\binom{r}{1}Q_{N_a1} + \binom{r}{2}Q_{N_a2} + \cdots + \binom{r}{r-1}Q_{N_ar-1}\right]\right\}(p-r)^{N\eta} \\
&= \binom{p}{r}Q_{N_ar}(p-r)^{N\eta}, \quad \text{where } Q_{N_ar} = r^{N_a} - \sum_{i=1}^{r-1}\binom{r}{i}Q_{N_ai}
\end{aligned} \tag{5.2}$$

Overall, the permutations where set R'_{rA} is disjoint with R_{BC} is θ_u, given below,

$$\theta_u = \sum_{r=1}^{N_a}\theta_{ur}$$

i.e.,

$$\left.\begin{aligned}
\theta_u &= \sum_{r=1}^{N_a}\binom{p}{r}Q_{N_ar}(p-r)^{N\eta} \\
\text{where} \quad Q_{N_ar} &= r^{N_a} - \sum_{i=1}^{r-1}\binom{r}{i}Q_{N_ai}, \text{ and } Q_{N_a1} = 1
\end{aligned}\right\} \tag{5.3}$$

This is the number of permutations in which there are $\leq N$ couplers between one of the sets, e.g. $R'rA$ and R_C. This condition will reveal exactly N private keys in R_{rA} associated with the N master keys, i.e. identifying node A as a traitor node.

Three Disjoint Sets

A traitor node is also found if all the sets R'_{rA}, R'_{rB} and R_C are disjoint. The number of permutations of these occurrences can be similarly found as follows.

(A). R_C has 1 Distinct Integer Consider that R_C uses one distinct integer, and the remaining integers are used in the other sets. The number of permutations is,

$$\begin{aligned} \theta_{c1} &= \binom{p}{1}[1]^{N_c} = \binom{p}{1} Q_{N_c 1} \\ \text{where} \quad Q_{N_c 1} &= [1]^{N_c} \end{aligned}$$

(a). R'_{rA} has 1 Distinct Integer The remaining $(p-1)$ integers can be used in set R'_{rA}. First consider that R'_{rA} uses only 1 distinct integer, while set R'_{rB} uses the remaining $(p-1-1)$ integers. The number of permutations is,

$$\begin{aligned} \theta_{x1_1} &= \binom{p-1}{1}[1]^{N_a}(p-1-1)^{N_b} = \binom{p-1}{1} Q_{N_a 1}(p-1-1)^{N_b} \\ \text{where} \quad Q_{N_a 1} &= [1]^{N_a} \end{aligned}$$

(b). R'_{rA} has 2 Distinct Integers If set R'_{rA} uses 2 distinct integers, the number of permutations for R'_{rA} and R'_{rB} is,

$$\theta_{x1_2} = \binom{p-1}{2}\left[2^{N_a} - \binom{2}{1}1^{N_a}\right](p-1-2)^{N_a} = \binom{p-1}{2} Q_{N_a 2}(p-3)^{N_b}$$

where

$$Q_{N_a 2} = \left[2^{N_a} - \binom{2}{1}1^{N_a}\right] = \left[2^{N_a} - \binom{2}{1} Q_{N_a 1}\right]$$

(c). R'_{rA} has 3 Distinct Integers Permutations if Set R'_{rA} uses 3 distinct numbers:

$$\begin{aligned}
\theta_{x1_3} &= \binom{p-1}{3}\left[3^{N_a} - \binom{3}{1}1^{N_a} - \binom{3}{2}\left\{2^{N_a} - \binom{2}{1}1^{N_a}\right\}\right](p-1-3)^{N_b} \\
&= \binom{p-1}{3}\left[3^{N_a} - \binom{3}{1}Q_{N_a1} - \binom{3}{2}Q_{N_a2}\right](p-1-3)^{N_b} \\
&= \binom{p-1}{3}Q_{N_a3}(p-4)^{N_b} \\
\text{where } Q_{N_a3} &= \left[3^{N_a} - \sum_{i=1}^{2}\binom{3}{i}Q_{N_ai}\right]
\end{aligned}$$

(d). R'_{rA} has N_a Distinct Integers If set R'_{rA} has N_a (all) distinct numbers, the number of permutations for R'_{rA} and R'_{rB} is,

$$\begin{aligned}
\theta_{x1_{N_a}} &= \binom{p-1}{N_a}\left[{N_a}^{N_a} - \binom{N_a}{1}1^{N_a} - \cdots\right](p-1-N_a)^{N_b} \\
&= \binom{p-1}{N_a}Q_{N_aN_a}(p-1-N_a)^{N_b} \\
\text{where } Q_{N_aN_a} &= \left[{N_a}^{N_a} - \binom{N_a}{1}1^{N_a} - \binom{N_a}{2}\left\{2^y - \binom{N_a}{1}1^{N_a}\right\} - \cdots\right] \\
&= {N_a}^{N_a} - \sum_{i=1}^{N_a-1}\binom{N_a}{i}Q_{N_ai}
\end{aligned}$$

Then for the case where R_C has only one distinct integer, the permutations possible are,

$$\theta_{d1} = \binom{p}{1}Q_{c_1} \times \sum_{i=1}^{N_a}\binom{p-1}{i}Q_{N_ai}(p-1-i)^{N_b}$$

(B). Two Distinct Couplers If R_C has two distinct numbers, the number of permutations is,

$$\theta_{c2} = \binom{p}{2}\left[2^{N_c} - \binom{2}{1}Q_{N_c1}\right] = \binom{p}{2}Q_{N_c2}$$

R'_{rA} can use $1, 2, \cdots, N_a$ distinct numbers giving permutations:

$$\begin{aligned}
\theta_{b2_1} &= \binom{p-2}{1}[1]^{N_a}(p-2-1)^{N_b} = \binom{p-2}{1}Q_{N_a1}(p-2-1)^{N_b} \\
\theta_{b2_2} &= \binom{p-2}{2}\left[2^{N_a} - \binom{2}{1}1^{N_a}\right](p-2-2)^{N_b} \\
&= \binom{p-2}{2}Q_{N_a2}(p-2-2)^{N_b} \\
&\vdots \\
\theta_{b2_{N_a}} &= \binom{p-2}{N_a}Q_{N_aN_a}(p-2-N_a)^{N_b} \\
\text{where } Q_{N_aN_a} &= {N_a}^{N_a} - \sum_{i=1}^{N_a-1}\binom{N_ai}{Q}_{N_ai}
\end{aligned}$$

Overall, the permutations for the case where there are 2 distinct integers in R_C are,

$$\theta_{d2} = \binom{p}{2}Q_{N_c2} \times \sum_{i=1}^{N_a}\binom{p-2}{i}Q_{N_ai}(p-2-i)^{N_b}$$

In general, if there are r integers in R_C, the number of permutations is,

$$\theta_{dr} = \binom{p}{r}Q_{N_cr} \times \sum_{i=1}^{N_a}\binom{p-r}{i}Q_{N_ai}(p-r-i)^{N_b}$$

Overall, the number of permutations for the case where there are $1, 2, \cdots, N$ distinct integers in R_C, where in each case there are $1, 2, \cdots, N_a$ distinct numbers in R'_{rA}, and R'_{rB} having the remaining unused integers is,

$$\left.\begin{aligned}
\theta_d = \sum_{r=1}^{N_c}\left[\binom{p}{r}Q_{N_cr} \times \sum_{k=1}^{N_a}\binom{p-r}{k}Q_{N_ak}(p-r-k)^{N_b}\right] \\
\text{where } \; Q_{N_cr} = r^{N_c} - \sum_{i=1}^{r-1}\binom{r}{i}Q_{N_ci} \\
\text{and } \; Q_{N_ak} = k^{N_a} - \sum_{i=1}^{k-1}\binom{k}{i}Q_{N_ai}
\end{aligned}\right\} \tag{5.4}$$

5.3.4 Probability of Finding a Traitor Node

Number of Traitor Node Permutations

The number of permutations where R'_{rA} is disjoint with R_{BC} is θ_u. Similarly, for the cases where R'_{rB} are disjoint with R_{AC}, there are also θ_u permutations. However, $2 \times \theta_u$ would double count the cases where R'_{rA}, R'_{rB} and R_C are all disjoint with each other. Hence the overall number of arrangements for; R'_{rA} disjoint with $(R'_{rB} \cup R_c)$ or, R'_{rB} disjoint with $(R'_{rA} \cup R_c)$ or, R'_{rA} disjoint with R'_{rB} disjoint with R_c is,

$$\theta_t = 2\theta_u - \theta_d \tag{5.5}$$

The total number of possible arrangements of p integers in sets R'_{rA}, R'_{rB} and R_C is $\left(p^{N\eta-N} \cdot p^{N\eta-N} \cdot p^{N}\right) = p^{2N\eta-N}$. Hence the probability of finding a traitor node is P_t given by,

$$\left.\begin{aligned} P_t &= \frac{2\theta_u - \theta_d}{p^{2N\eta-N}} \\ &\text{where,} \\ \theta_u &= \sum_{r=1}^{N_a} \binom{p}{r} Q_{N_a r} (p-r)^{N\eta} \\ \theta_d &= \sum_{r=1}^{N_c} \left[\binom{p}{r} Q_{N_c r} \times \sum_{k=1}^{N_a} \binom{p-r}{k} Q_{N_a k} (p-r-k)^{N_b} \right] \\ Q_{N_a r} &= r^{N_a} - \sum_{i=1}^{r-1} \binom{r}{i} Q_{N_a i} \;\text{ and }\; Q_{N_c r} = r^{N_c} - \sum_{i=1}^{r-1} \binom{r}{i} Q_{N_c i} \end{aligned}\right\} \tag{5.6}$$

The probabilities P_t for keying parameters are given in Table. (5.1).

Simulation of Probabilities

To check the correctness of Eqns. (5.3), (5.4), and (5.6), a MATLAB programme was written and used to simulate probabilities for different cases. This is given in Appendix B.1. First, set R_C is filled with N random integers $\in [0, p-1]$. Then sets R'_{rA} and

η	N	Probability of finding a Traitor Node				
		$p = 13$	$p = 17$	$p = 19$	$p = 23$	$p = 31$
6	6	1.07×10^{-16}	1.68×10^{-15}	6.25×10^{-15}	7.41×10^{-14}	5.60×10^{-12}
	7	5.23×10^{-20}	8.71×10^{-190}	3.39×10^{-18}	4.59×10^{-17}	5.04×10^{-15}
	8	2.54×10^{-23}	4.37×10^{-22}	1.75×10^{-21}	2.58×10^{-20}	3.72×10^{-18}
7	6	2.42×10^{-20}	4.05×10^{-19}	1.59×10^{-18}	2.17×10^{-17}	2.47×10^{-15}
	7	2.89×10^{-24}	5.04×10^{-23}	2.03×10^{-22}	3.05×10^{-21}	4.68×10^{-19}
	8	3.47×10^{-28}	6.12×10^{-27}	2.51×10^{-26}	4.00×10^{-25}	7.54×10^{-23}
8	6	5.46×10^{-24}	9.49×10^{-23}	3.82×10^{-22}	5.74×10^{-21}	8.68×10^{-19}
	7	1.62×10^{-28}	2.86×10^{-27}	1.18×10^{-26}	1.88×10^{-25}	3.61×10^{-23}
	8	4.81×10^{-33}	8.59×10^{-32}	3.57×10^{-31}	5.92×10^{-30}	1.32×10^{-27}

Key sizes 64 bits, 80 bits, 96 bits, 112 bits, 128 bits

Table 5.1: Probabilities of Finding a Traitor Node

R'_{rB} are filled with $(N_a = N\eta - N)$ random integers. Sets $R_{BC} = (R'_{rB} \cup R_C)$ and $R_{AC} = (R'_{rA} \cup R_C)$ are formed. Then set R'_{rA} is compared with (R_{BC}) and R'_{rB} is compared with (R_{AC}) and are counted if they intersect for each run. By dividing the total number of counts when they intersect with the total number of runs, the probability of finding P_t can be found. Some runs take an extremely long time. The results for reasonable number runs up to 10^{12} compare very well with the Eqn. (5.6), as shown in Table (5.2).

The recursive expressions for Q_{N_ar} and Q_{N_cr} in Eqn. (5.6) can be efficiently computed if they are pre-calculated and then used to compute θ_d and θ_u. The Linux Genius Mathematical Tool code to compute P_t as in Eqn. (5.6) is given in Appendix B.1.

N	η	$p = 7$		$p = 13$		$p = 1023$	
		P_t	Simulation	P_t	Simulation	P_t	Simulation
2	2	0.47016	0.4702	0.6951	0.6948	0.9961	0.9961
	3	0.06995	0.0699	0.2450	0.2451	0.9844	0.9845
3	2	0.14496	0.1450	3.8865×10^{-1}	3.887×10^{-1}	0.9912	0.9912
	3	3.606×10^{-3}	3.600×10^{-3}	3.5108×10^{-2}	3.510×10^{-2}	0.96521	0.9653
	4	6.008×10^{-5}	6.160×10^{-5}	1.3587×10^{-3}	1.400×10^{-3}	0.9234	0.9237
4	3	1.374×10^{-4}	1.304×10^{-4}	2.7208×10^{-3}	2.700×10^{-3}	0.93865	0.9385
	4	4.934×10^{-7}	5.3182×10^{-7}	1.9545×10^{-5}	1.9539×10^{-5}	0.8673	0.8679

Table 5.2: Comparing P_t with Simulation Results

5.3.5 Node Capture to Find a Traitor Node

If the adversary is able to capture any number of nodes and repeatedly carry out the pairing attacks, eventually a traitor node will be found. This attack should be done recursively to minimise the number of nodes required. As a new node is captured, it is paired with each of the previously captured nodes to find a traitor node.

The number of nodes that need to be captured can be estimated. For given keying parameters of N, η and p, the probability of finding a traitor node P_t can be computed from Eqn. (5.6). The expected number of attempts to find one occurrence is then $\frac{1}{P_t}$. Each node has η public keys, so each pairing allows η^2 attempts. The number of expected pairs of nodes required is then reduced to $\frac{1}{P_t} \times \frac{1}{\eta^2} = \frac{1}{P_t \eta^2}$. If the number of nodes required to be captured is n_c then, the number of pairs that are required to be formed is $\binom{n_c}{2}$, i.e.

$$\begin{aligned} & \frac{n_c!}{2!(n_c-2)!} \geqslant \frac{1}{\eta^2 P_t} \\ \text{i.e.} \quad & \frac{n_c(n_c-1)}{2} \geqslant \frac{1}{\eta^2 P_t} \\ \text{giving} \quad & n_c \geqslant \frac{1}{2}\left(1 + \sqrt{1 + \frac{8}{\eta^2 P_t}}\right) \end{aligned} \tag{5.7}$$

The number of nodes required to be captured are shown in Table (5.3) for some parameters with $p = 13 \sim 31$ and $N, \eta = 6, 7, 8$. It can be seen that with suitable

values, the probabilities are extremely small and thousands of nodes need to be captured.

Effort Required

As each node is captured and paired with all the previous nodes to find the traitor node, the number of pairings increase as an arithmetic progression. Using n_c captured nodes, if the attempt is successful only at the last pairing, the total number of pairing operations is then,

$$\Theta_p = \sum_{u=1}^{n_c-1} u = \frac{1}{2} n_c(n_c - 1) \tag{5.8}$$

Each pairing involves $N\eta^2$ multiplication of $N\eta$ $(m\times 1)$ row vectors with η $(1\times m)$ column vectors in each node, and comparing the results each time. Just counting the multiplication operations, there are $2\times mN\eta^2$ operations. The number of multiplication operations to find a traitor node is then,

$$\Theta_m = n_c(n_c - 1)mN\eta^2 \tag{5.9}$$

If the capture size is $n_c = 10,000$, $N = 12$, $\eta = 4$, $m = 16$, then $\Theta_m = 3.07\times 10^{13}$. While this number is large, it is feasible using a very powerful computer.

5.3.6 Use of the Traitor Node

Finding a traitor node does not break the scheme but only improves the chances of finding the PPMka in subsequent pairings. As shown in Fig. (5.8), a node B paired with the traitor node must have exactly N couplers in order to distinctly reveal its PPMka. The probability of finding this in node B is the same as finding the traitor node itself.

It can be seen that to discover the PPMka by finding a pair of nodes in which one of them would expose their PPMka requires a large number of nodes if suitable keying parameters are used. This is due to the operations over a small field $\mathbb{F}_p$. The small

η	N	Values of prime, p				
		13	17	19	23	31
4	4	8.05×10^{1}	3.27×10^{1}	2.27×10^{1}	1.25×10^{1}	5.6
	5	7.72×10^{2}	2.64×10^{2}	1.66×10^{2}	7.49×10^{1}	2.32×10^{1}
	6	7.98×10^{3}	2.43×10^{3}	1.43×10^{3}	5.57×10^{2}	1.29×10^{2}
	7	8.55×10^{4}	2.41×10^{4}	1.35×10^{4}	4.70×10^{3}	8.45×10^{2}
	8	9.34×10^{5}	2.49×10^{5}	1.34×10^{5}	4.31×10^{4}	6.32×10^{3}
5	4	9.07×10^{2}	3.03×10^{3}	1.88×10^{2}	8.27×10^{1}	2.45×10^{1}
	5	1.89×10^{4}	5.51×10^{3}	3.16×10^{3}	1.16×10^{3}	2.38×10^{2}
	6	4.12×10^{5}	1.11×10^{5}	6.02×10^{4}	1.96×10^{4}	3.01×10^{3}
	7	9.16×10^{6}	2.35×10^{6}	1.23×10^{6}	3.67×10^{5}	4.51×10^{4}
	8	2.05×10^{8}	5.10×10^{7}	2.61×10^{7}	7.31×10^{6}	7.58×10^{5}
6	4	1.16×10^{4}	3.42×10^{3}	1.97×10^{3}	7.37×10^{2}	1.56×10^{2}
	5	5.07×10^{5}	1.35×10^{5}	7.30×10^{4}	2.34×10^{4}	3.49×10^{3}
	6	2.28×10^{7}	5.74×10^{6}	2.98×10^{6}	8.66×10^{5}	9.96×10^{4}
	7	1.03×10^{9}	2.53×10^{8}	1.28×10^{8}	3.48×10^{7}	3.32×10^{6}
	8	4.68×10^{10}	2.85×10^{10}	1.42×10^{10}	3.66×10^{9}	2.95×10^{8}
7	4	1.16×10^{4}	3.42×10^{3}	1.97×10^{3}	7.37×10^{2}	1.56×10^{2}
	5	5.07×10^{5}	1.35×10^{5}	7.30×10^{4}	2.34×10^{4}	3.49×10^{3}
	6	2.28×10^{7}	5.74×10^{6}	2.98×10^{6}	8.66×10^{5}	9.96×10^{4}
	7	1.03×10^{9}	2.53×10^{8}	1.28×10^{8}	3.48×10^{7}	3.32×10^{6}
	8	4.68×10^{10}	2.85×10^{10}	1.42×10^{10}	3.66×10^{9}	2.95×10^{8}
	8	1.09×10^{13}	2.58×10^{12}	1.27×10^{12}	3.20×10^{11}	2.33×10^{10}
8	4	2.24×10^{6}	5.79×10^{5}	3.06×10^{5}	9.32×10^{4}	1.22×10^{4}
	5	4.11×10^{8}	1.01×10^{8}	5.12×10^{7}	1.40×10^{7}	1.37×10^{6}
	6	7.57×10^{10}	1.82×10^{10}	9.04×10^{9}	2.33×10^{9}	1.90×10^{8}
	7	1.39×10^{13}	3.30×10^{12}	1.63×10^{12}	4.07×10^{11}	2.94×10^{10}
	8	2.55×10^{15}	6.03×10^{14}	2.96×10^{14}	7.26×10^{13}	4.86×10^{12}

Key Sizes 64 bits, 80 bits, 96 bits, 112 bits, 128 bits

Table 5.3: Capture Sizes to find a Traitor Node

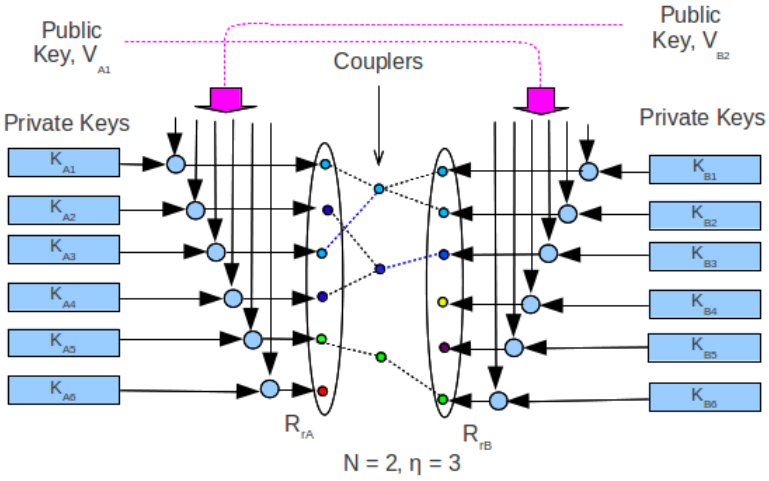


Figure 5.10: Pairing Attack for Case with $N = 2, \eta = 3$

p also means that each of the key-set integers is small. However, due to the use of multiple keys in permutations, there are $N\eta^2$ integers making it possible to construct large pairwise keys of 64, 80, 112, 128 and even 192 bits. On the other hand, if p is large, for example 16 bits, the probability of collisions is small and the pairing attack will quickly expose the PPMka of the private keys.

5.4 Pairing Attack with Limited Capture

In the pairing attack, using one of the η public keys say $\mathbf{V}_{A1}$ with node B, we obtain the partial key-sets R_{rA} and R_{rB}, each with $N\eta$ elements in $[0, p-1]$, see Fig. (5.10). Let node A have N_{c_i} couplings, each one possibly linking the related private key say $\mathbf{K}_{A_x}$ to one of the master keys, say $\mathbf{M}_1$, and the public key say, $\mathbf{V}_{A_1}$ that is used. These linkages can be used to form the equation $\mathbf{K}_{A_x} = \mathbf{V}_{A_1}^T \mathbf{M}_1$. Since each one is as likely to be correct, there are N_{c_i} ways to do this. This is smaller than the brute force method where there are $N\eta$ possibilities. Next, using another public key say $\mathbf{V}_{A2}$, N_{c_2} couplings are obtained, giving N_{c_2} possible equations, and so on. By using all the η public keys

for the pairing, the total number of equations to solve for $\mathbf{M}_1$ is,

$$\Phi_{1_u} = \prod_{u=1}^{\eta} N_{c_u} \tag{5.10}$$

This process is repeated for $\frac{m}{\eta}$ nodes to obtain $(m \times m)$ equations for solving $\mathbf{M}_1$ and the total number of possible solutions is,

$$\Phi_1 = \prod_{v=1}^{\frac{m}{\eta}} \left(\prod_{u=1}^{\eta} N_{c_{u_v}} \right) \tag{5.11}$$

The number of couplings obtained in each pairing is N_{c_i} varies for each run, but for simplicity, if the mean value is N_c, then Eqn. (5.11) can be simplified to,

$$\Phi_1 = [N_c]^{\frac{m}{\eta}\eta} = [N_c]^m \tag{5.12}$$

After solving for $\mathbf{M}_1$, the associated private keys can be removed and the remaining keys used to solve for $\mathbf{M}_2, \cdots, \mathbf{M}_N$. The total number of possible solutions is then

$$\Phi = \sum_{i=0}^{m} [N_c - i]^m \tag{5.13}$$

5.4.1 Binomial Distribution Approximation

Fig. (5.11) shows the distribution of the number of couplings using a simulation of the pairing attacks for the case $p = 31, N = 6, \eta = 6$. It suggests that the distribution can be approximated by the binomial distribution,

$$P(X = x) \;=\; \binom{N\eta}{x} p_r^x (1 - p_r)^{(N\eta - x)} \tag{5.14}$$

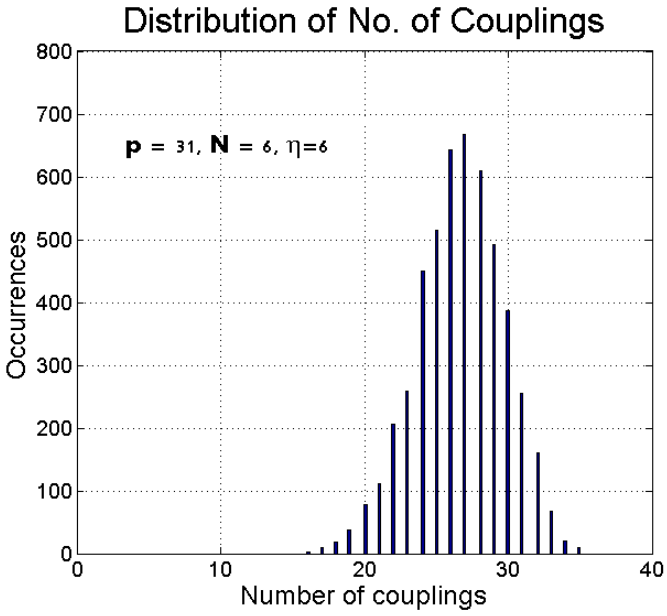


Figure 5.11: Distribution of the Number of Couplings for $p = 31, N = 6, \eta = 6$

The probability p_r can be found by using $P(X = N) = P_t$ from Eqn. (5.6), i.e.,

$$P_t = \binom{N\eta}{N} p_r^N (1 - p_r)^{(N\eta - N)} \tag{5.15}$$

The mean of the binomial distribution is given by,

$$\mu = N\eta p_r \tag{5.16}$$

If we let the expected number of couplings in a pairing be $N_c = \mu$, then the number of iterations required is,

$$\Phi = \sum_{i=0}^{N-1} [N_c - i]^m = \sum_{i=0}^{N-1} [\mu - i]^m \tag{5.17}$$

Table (5.4) gives the probable number of master keys solutions in $log_{10}(\Phi)$, for various keying parameters. The quantity Φ represents the number of iterations required to solve for the master keys and, with suitable parameters, can be made as large as

desired. For example with $m = 24, p = 31, N = 8, \eta = 8$, $\Phi = 10^{40.57} = 2^{134}$.

5.5 Security Strength of the BYka Scheme

In the above section §5.3.4, to find a traitor node, the effort and number of steps to capture, extract, compute and compare, can be very large, and it would require n_c nodes to be captured. A traitor node, if available, does not break the scheme but makes it easier to find the PPMk of other nodes. If the number of nodes in the network is less than n_c, then the most efficient way to break the scheme is the limited capture pairing attack described above in §5.4. It results in the least amount of ambiguities to associate the private key with the master key and public key used. The total number of trials required to derive the master keys is Φ. Each trial consists of solving the system of equations, testing the master key, as detailed in §4.3.4 which includes at least 10^3 multiplication operations for $m \geq 16$. In NIST (Barker et al., 2012), the definition of security strength is the number of operations required to break a scheme, see §1.3.3. To be very conservative, consider that each trial is just one operation. Then the security strength of the BYka scheme is given by Φ. By selecting appropriate keying parameters, it can be $64, 80, 112, 128$ or 192 bits.

Definition 10 (BYka Scheme Security Strength) *If the network size is less than the traitor node capture size, then the security strength of the BYka scheme is given by Φ, the number of trials of solutions of the system of equations to derive the master keys, given by Eqn. (5.17).*

5.6 Summary

The private-public-master-key-association (PPMka) information is crucial for breaking the underlying Blom's scheme. The PPMka cannot be found by examining the node and

Values of $log\Phi$, Probable number of master key solutions Φ										
η	N	$p = 13$			$p = 17$			$p = 31$		
m		12	16	24	12	16	24	12	16	24
3	3	8.39	11.18	16.78	8.39	11.18	16.78	7.22	9.63	14.45
	4	10.84	14.45	21.67	10.84	14.45	21.67	9.34	12.45	18.68
	5	12.50	16.66	24.99	12.50	16.66	24.99	11.45	15.27	22.90
	6	13.75	18.34	27.51	13.75	18.34	27.51	12.95	17.27	25.90
	7	14.77	19.69	29.53	14.45	19.27	28.90	13.75	18.34	27.51
	8	15.61	20.82	31.23	15.35	20.46	30.69	14.77	19.69	29.53
4	3	10.84	14.45	21.67	10.14	13.52	20.28	9.34	12.45	18.68
	4	12.95	17.27	25.90	12.50	16.66	24.99	11.45	15.27	22.90
	5	14.11	18.82	28.23	14.11	18.82	28.23	13.37	17.82	26.74
	6	15.35	20.46	30.69	15.06	20.08	30.13	14.45	19.27	28.90
	7	16.34	21.79	32.68	16.11	21.48	32.22	15.61	20.82	31.23
	8	16.98	22.64	33.96	16.98	22.64	33.96	16.56	22.08	33.13
5	3	12.00	16.00	24.00	12.00	16.00	24.00	10.84	14.45	21.67
	4	14.11	18.82	28.23	13.75	18.34	27.51	12.95	17.27	25.90
	5	15.35	20.46	30.69	15.35	20.46	30.69	14.45	19.27	28.90
	6	16.56	22.08	33.13	16.34	21.79	32.68	15.87	21.16	31.73
	7	17.37	23.16	34.73	17.37	23.16	34.73	16.78	22.37	33.55
	8	18.22	24.30	36.44	18.06	24.08	36.12	17.73	23.63	35.45
6	3	13.37	17.82	26.74	12.95	17.27	25.90	12.00	16.00	24.00
	4	15.06	20.08	30.13	14.77	19.69	29.53	14.11	18.82	28.23
	5	16.34	21.79	32.68	16.34	21.79	32.68	15.61	20.82	31.23
	6	17.55	23.40	35.10	17.37	23.16	34.73	16.98	22.64	33.96
	7	18.38	24.50	36.76	18.22	24.30	36.44	17.90	23.86	35.79
	8	19.09	25.46	38.19	19.09	25.46	38.19	18.68	24.90	37.35
7	3	14.11	18.82	28.23	13.75	18.34	27.51	12.95	17.27	25.90
	4	15.87	21.16	31.73	15.87	21.16	31.73	15.06	20.08	30.13
	5	17.18	22.90	34.35	17.18	22.90	34.35	16.56	22.08	33.13
	6	18.22	24.30	36.44	18.22	24.30	36.44	17.73	23.63	35.45
	7	19.09	25.46	38.19	19.09	25.46	38.19	18.68	24.90	37.35
	8	19.95	26.60	39.91	19.84	26.45	39.68	19.48	25.97	38.96
8	3	14.77	19.69	29.53	14.77	19.69	29.53	13.75	18.34	27.51
	4	16.56	22.08	33.13	16.56	22.08	33.13	15.87	21.16	31.73
	5	17.90	23.86	35.79	17.90	23.86	35.79	17.37	23.16	34.73
	6	18.96	25.28	37.92	18.82	25.09	37.64	18.53	24.71	37.06
	7	19.84	26.45	39.68	19.72	26.30	39.44	19.48	25.97	38.96
	8	20.59	27.46	41.18	20.49	27.32	40.98	20.28	27.04	40.57

Key sizes 64 bits, 80 bits, 96 bits, 112 bits, 128 bits

Table 5.4: Probable Number of Master Key Solutions Assuming $N_c = \mu$, in $log(\Phi)$

its keys, and the brute force attempt would require an infeasible amount of resources and time. However, if a pair of nodes are taken, and using each other's public keys with their private keys to compute their pairwise key-sets R results in exactly N unique identical integers across both sets, then the PPMka of the related private keys are exposed. This is not possible in the BYka scheme where the key computations are over a very small e are exactly N unique identical integers shared in both sets, then the PPMka of the related privafield $\mathbb{F}_p$ and p is a prime ≤ 31. Most likely, there are more than N unique identical integers shared across the two sets, making the PPMka ambiguous.

An efficient approach is to use the pairing attack where only one of the nodes' public keys is used to compute the partial key-sets in each node. Now, there are only $N\eta$ integers in these key-sets, increasing the chance of having only unique integers. However, due to the small number of integers available, there will be ambiguities. To make headway, a traitor node, where all the private keys related to one of the master keys are known, is required. If this node is found, it does not break the scheme, but subsequent pairings will be slightly easier. The analytical results were obtained to estimate the probability of finding a traitor node and the number of nodes to capture to find one. This showed that using suitable keying parameters, this probability can be made so small that it would require tens of thousands of nodes to be captured. If the network size is smaller than this, then the PPMka cannot be exposed using this attack.

Alternatively, without finding a traitor node, the pairing attack produces the smallest number of elements in the partial key-sets. This results in the smallest number of solutions Φ possible, and is much smaller than the brute force method. Using suitable parameters, the analytical results showed that Φ can be made extremely large requiring 2^{64} up to 2^{192} solutions. This means the BYka scheme is able to achieve security strengths of 64, 80, 112, 128, and 192 bits using suitable keying parameters.

The analysis showed that due to the lack of the PPMka information, the BYka scheme can be used in networks of any desired size up to millions of nodes. The BYka

scheme is secure in terms of the strengths of all the keys used, the underlying Blom's scheme cannot be broken even if the entire network of nodes was captured, and the effort required to break it can be designed to meet adequate security strengths in the NIST recommendations.

Chapter 6

Evaluation and Performance

6.1 Introduction

The previous chapters showed that the Blom's scheme can be modified so that it is secure against a large number of nodes being compromised. The analytical results obtained enable predictions of the number of nodes that need to be captured in order to discover the PPMka, and the number of trials Φ required to find the master keys. These results are now verified by conducting some experiments.

The second section shows how the BYka scheme developed in this thesis can be implemented in real sensor nodes. This is done using the MICAz mote to demonstrate its practicality and to obtain some information on the computation times for comparison with other schemes.

6.2 Experiment – Attacks to Obtain the PPMka

The aim of the experiment is to mount the pairing attacks on the implementations of the BYka scheme to find traitor nodes and the number of possible master key solutions Φ for various keying parameters. These will be compared with theoretical values obtained

in the previous chapter.

The BYka scheme was implemented using a computer program running MATLAB. The computer hardware used was an i5-2500 - 3.3 GHz dual core server with 16 GB RAM. The software used was MATLAB R2012b running in Windows Server 2008 R2 Datacenter. Using the given keying parameters, N, η, m, p, and q and its built-in random number generator, the program acting as TA, generates its master keys. It creates a node by generating a new unique random *ID* $s \in [0, q-1]$ satisfying Eqn. (3.6), and computes the private keys using Eqn. (3.3). The keying parameters, *ID*s and private keys can be transferred to the sensor device using a cable and then deployed in a real implementation.

6.2.1 Simulation of Attacks

In these experiments, the capturing of nodes and extracting their keying material are simulated in the computer program by simply storing the nodes into a "pool" of captured nodes. This greatly speeds up the experiment without any loss of realism. Real life attacks involving physically removing the nodes and extracting their keys would take much longer and while being more realistic, would not contribute to any better result.

Computer Program

The main steps for the program are shown in Fig. (6.1), and the code written using MATLAB is given in Appendix (B.2). The programme, using the given keying parameters N, η, m, p, and q, first generates the N master keys over the prime field $\mathbb{F}_p$. Then it creates a node by selecting a random ID_A complying with Eqn(3.6) and computes the public keys and the corresponding private keys. This node is then put into the capture pool of nodes collectively called "nodes A". Next, it generates a new node B such that its *ID* is new and not in the pool. Then, taking one node A from the pool, and for each

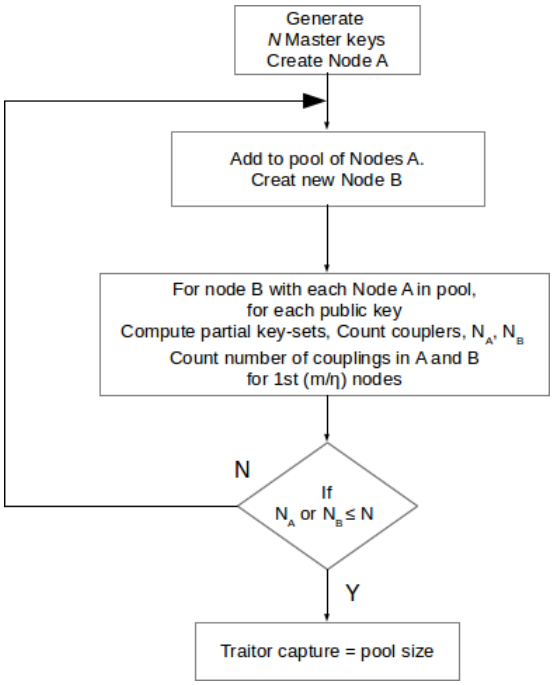


Figure 6.1: Attack on PPMka

public key in node B and A, the partial key-sets are computed and compared to identify the couplers (identical integers across the two key-sets). The numbers of couplers in each key-set are counted and if either set has $\leq N$ couplers, a traitor node is found. If not, node B is added to the capture pool of nodes A. A new node B is created and the pairing attack repeated. At the same time, the number of couplings in nodes A and B are accumulated for the first $\frac{m}{\eta}$ nodes. This simulates capturing just sufficient of nodes to obtain the required sets of equations to solve for the master keys.

When a traitor node is found, a new set of master keys is generated and the process repeated to obtain 1000 values of traitor capture sizes using the same keying parameters of N, η, p and q. The experiments were repeated for various keying parameters. As some runs can take an extremely long time, measured in months and years using the available computing resources, the parameters are chosen so that results can be obtained within reasonable times.

6.2.2 Experimental Results

Execution Times

Each pairing attack requires computations of the partial key-set involving multiplications of one $(1\times m)$ row vector and $N\eta$ $(m\times 1)$ column vectors. Just considering the multiplication operations, there are $mN\eta$ operations. For one pair of nodes, there are η keys to use with each other, giving a total of $mN\eta\times\eta\times\eta = mN\eta^3$ multiplications. As a new node is "captured", it is paired with the each of the previous nodes recursively until the traitor node is found. If n_c nodes are captured and the traitor node is found at the last possible pairing, $\sum_{i=1}^{n_c}(i-1)$ comparisons need to be made. The computation time for one run is approximately,

$$T_a \quad \propto \quad \sum_{i=1}^{n_c}(i-1)mN\eta^3 = \frac{1}{2}n_c(n_c-1)mN\eta^3$$

As an indication, for the case $m = 24, p = 31, \eta = 4, N = 5$ with 1000 runs the MATLAB script took $32,899$ seconds capturing an average of 21.48 nodes. Using this result to obtain the proportional constant, the computation time per run is approximately,

$$T_{n_c} \approx 1.623 \times 10^{-5}(n_c-1)n_cmN\eta^3 \quad \text{seconds}$$

Some parameters, for example with $p = 31,\ m = 24,\ N = 6$ and $\eta = 6$, the expected traitor capture size is $> 99,000$ and to find a traitor node can take approximately over 150 years using our system.

Confidence Interval for the Means

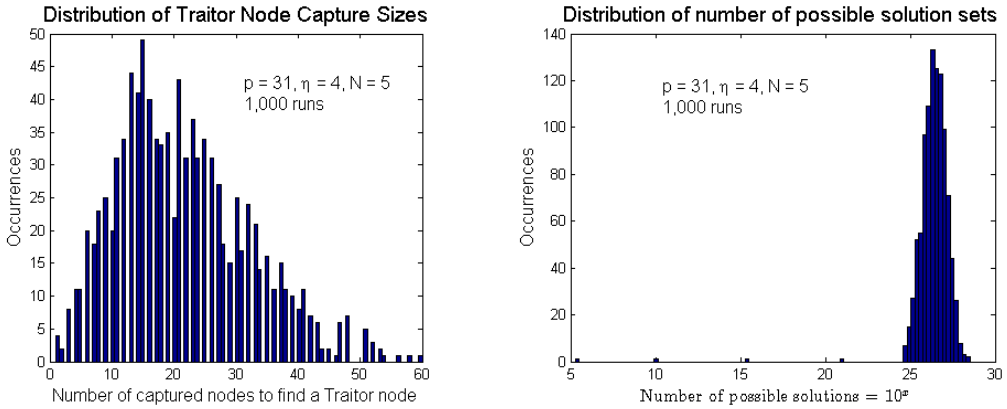
The confidence intervals for the number of solutions of master keys Φ were computed as this is an important quantity related to the security strength. In our experiments, the samples sizes (runs) are ≥ 100. It is assumed that the runs are independent and the distribution of errors in the runs can be approximated as a normal distribution. For 95% confidence interval, the critical value $z = 1.96$ was used, and the confidence interval is given by, $\bar{x} \pm 1.96\frac{s}{\sqrt{n}}$.

Traitor Capture Sizes

Table (6.1) shows the traitor node capture sizes n_c for various keying parameters which are able to yield results within a reasonable time.

Comparison with Theory Fig. (6.2) shows the typical distribution of the results of pairing attacks over 1000 runs for the simple case $m = 24, p = 31, \eta = 4, N = 5$. In Fig (6.2(a)), the mean was 21.48 captured nodes. The estimated capture size from Eqn. (5.6) and Table (6.1) is 23 nodes which is larger. The capture sizes in the experiments are consistently smaller than the theoretical values. This may be explained by differences between the way a node is created in the theoretical and experimental cases.

Rejected Nodes The theoretical estimated capture size assumes that the node *ID*s are uniformly distributed over $\mathbb{F}_q$ and as it simply counts the number of attempts required, it also allows for a newly captured node to have the same *ID* as those previously captured. In addition, it allows for nodes with malformed *ID*s, i.e. those not complying with Eqn. (3.6). On the other hand, in the experiments, all nodes have unique *ID*s to reflect real systems, and there are no malformed *ID*s. As a node is created if it has a malformed *ID*, or is identical to any node in the pool, it is "rejected" and not added to the pool. This does not happen in the theoretical case. As the number of captured



(a) Traitor Capture Sizes (b) Number of Solution Sets

Figure 6.2: Result of Pairing Attacks on the BYka Scheme Using $m = 24, p = 31, \eta = 4, N = 5$

nodes in the experiment increases, the number of rejected nodes increases. As a result the number of nodes captured in the experiment would be always less than the number given by Eqn. (5.7).

Estimate of Rejected Nodes An *ID* is malformed if $s^{m-1} < q$. For $m = 24, q = 65521$, an *ID* can take all values except for 0 and 1. Another condition is that $s^{m-1} \equiv r \pmod{p}$, and $r \neq 0$ or $r \neq s$. This condition is easily met and the number of malformed *ID* are ignored for this estimate.

In the experiments, if the current captured pool size is n_c, then the probability of getting another node with the same *ID* as any one in the pool is $n_c(\frac{1}{q})$. The total probability that a node has the same *ID* as one in the pool when the sizes were $\{1, 2, 3, \cdots, n_c\}$, is $\sum_{i=1}^{n_c} \frac{i}{q} = \frac{n_c(n_c+1)}{2q}$.

If the total number of nodes that would have been captured without any rejection is n_t, then ,

$$n_t = n_c + \frac{n_c(n_c+1)}{2q} \times n_c$$

Line	η	N	Traitor Capture size n_c		Include rejects		% Diff of estimated
			Estimated	Expt.	n_r	Expt.+ n_r	
1	4	4	5.59	5.23	0.001	5.231	6.9
2	6	3	10.76	9.62	0.008	9.628	11.2
3	4	5	23.23	21.48	0.079	21.559	8.1
4	5	4	24.45	21.37	0.078	21.448	14.0
5	7	3	37.57	33.04	0.284	33.324	12.7
6	4	6	128.05	113.53	11.264	124.795	2.6
7	6	4	155.91	135.88	19.281	155.166	0.5
8	5	5	237.99	209.22†	70.221	279.441	−14.8†

Table 6.1: Comparison Between Analytical and eExperimental Results for $m = 24,\ p = 31$, †600 Runs only as it took too long

Let the number of rejected nodes be n_r, i.e. $n_t = n_c + n_r$. Then

$$\begin{aligned} n_c + n_r &= n_c + \frac{n_c(n_c+1)}{2q} \times n_c \\ n_r &= \frac{n_c^2(n_c+1)}{2q} \end{aligned} \tag{6.1}$$

The expected rejected nodes n_r are added to the experimental capture sizes for comparison. This is shown in Table (6.1), arranged in ascending order of n_c for clarity. For small values of n_c, lines 1 to 5, the percentage difference between the theoretical and experimental n_c is quite large, up to about 14%. Since n_c is small, less than about 33 nodes, the estimates of the rejected nodes may be quite inaccurate. However, for cases where n_c are larger between 100 to 140 nodes in lines 6 and 7, the % differences are less than 2.6% of the theoretical value. Line 8 is an anomaly with an error that is about 14.8%. This may be due to the capture size being obtained for 600 runs only because of the long execution time, instead of 1000 in other cases.

η	N	Calculated Φ	Experimental Φ with 95% confidence			
			Φ_{mean}	std, s	Φ_{min}	Φ_{max}
4	4	2.61×10^{23}	2.08×10^{24}	2.1013×10^{24}	1.95×10^{24}	2.21×10^{24}
	5	5.87×10^{26}	2.34×10^{27}	1.8598×10^{27}	2.23×10^{27}	2.46×10^{27}
	6	2.20×10^{29}	7.32×10^{29}	5.2156×10^{29}	7.00×10^{29}	7.65×10^{29}
5	4	1.69×10^{26}	1.85×10^{27}	9.7374×10^{26}	1.79×10^{27}	1.91×10^{27}
	5	2.67×10^{29}	1.39×10^{30}	8.8243×10^{29}	1.33×10^{30}	1.44×10^{30}
	6	7.87×10^{31}	4.55×10^{32}	2.6604×10^{32}	4.39×10^{32}	4.72×10^{32}
6	4	3.06×10^{28}	2.47×10^{29}	1.9004×10^{29}	2.36×10^{29}	2.59×10^{29}
	5	3.71×10^{31}	3.41×10^{32}	2.3599×10^{32}	3.26×10^{32}	3.56×10^{32}
	6	9.13×10^{33}	1.09×10^{35}	5.3238×10^{34}	1.05×10^{35}	1.12×10^{35}
7	4	2.33×10^{30}	2.78×10^{31}	1.8956×10^{31}	2.67×10^{31}	2.90×10^{31}
	5	2.89×10^{33}	3.82×10^{34}	2.0717×10^{34}	3.69×10^{34}	3.95×10^{34}
	6	4.29×10^{36}	1.14×10^{37}	5.1243×10^{36}	1.11×10^{37}	1.17×10^{37}

Table 6.2: Comparison Between Analytical and Experimental Results for Φ using $m = 24,\ p = 31$

Number of Solutions Φ

Fig. (6.2(b)) shows the distribution of the number of possible of solutions for pairing attacks over 1000 runs for the simple case $m = 24, p = 31, \eta = 4, N = 5$. The mean was 8.467×10^{26} compared to 5.428×10^{26} computed from Eqn. (5.15). Interestingly, there were 4 cases where the number of possible solutions were below 10^{20}, with one case with only $187,000$ possibilities. This particular case with 1000 runs took over 9 hours on our system.

Table (6.2) shows the comparison between the theoretical estimated and experimental values of Φ. The estimated values are based on approximating the couplings as a binomial distribution, Eqn. (5.15), (5.17). The mean μ was calculated by solving Eqn. (5.15) to obtain the probability p_r. The experimental values of Φ, are obtained as the product of $N_{c_{1,\cdots,m}}$ where N_c are the number of couplings obtained in each pairing. The values of Φ, which are the number of sets of master key solutions, are consistently

Security Strength	Calculated	Expt Φ 95% confidence			Parameters			
	Φ	Φ_{mean}	Φ_{min}	Φ_{max}	p	m	η	N
192	1.00×10^{58}	3.03×10^{58}	2.92×10^{58}	3.14×10^{58}	61	38	4	12
128	9.04×10^{38}	1.68×10^{41}	1.64×10^{41}	1.72×10^{41}	31	24	5	11
112	3.72×10^{35}	2.33×10^{36}	2.28×10^{36}	2.38×10^{36}	31	24	4	11
80	8.19×10^{24}	1.84×10^{25}	1.80×10^{25}	1.88×10^{25}	23	24	3	12
64	2.06×10^{19}	8.47×10^{18}	8.36×10^{19}	8.58×10^{19}	31	24	3	12

Table 6.3: Comparison Between Calculated and Experimental Security Strength. Φ is Number of Trials Required

larger than the calculated values by a factor of about 10. The 95% confidence level for Φ_{mean} was calculated using $(\Phi_{mean} \pm 1.96\frac{s}{\sqrt{n}})$ where s is the standard deviation and n the sample size (runs). They showed that the theoretical estimated values are conservative approximations.

Security Strength The security strength is measured as the number of operations to break the cryptosystem using a most efficient algorithm, see §1.3.3. The pairing attack using one public key at a time gives the least number of possible solutions Φ. If we consider each operation as one solution for the master key, testing it, etc., as detailed in §4.3.4, Φ can be considered (very conservatively) as the security strength for the BYka scheme. Table (6.3) shows the various security strengths based on the theoretical estimated values which are more conservative than the experimental ones.

6.3 Hardware Implementation

The BYka scheme was implemented in a commercial sensor device to demonstrate its practicality as well as to determine the pairwise key computation times for comparison with other schemes. This thesis is not about implementation and no effort was made to optimise, calibrate, or to evaluate the operational aspects in real networks.

6.3.1 Hardware and Software Platforms

The MICAz mote (Memsic Corp., 2012) is chosen as the hardware platform in this study so that the results can be meaningfully compared to those obtained by other researchers who often use this device as well. The MICAz sensor mote hardware consists of an 8-bit ATmega 128L μController @ 8 MHz, with 4 KB EEPROM, 128 KB Flash ROM, and 4 MB RAM. The on-board C2420 chip has an IEEE 802.15.4 radio transceiver and an AES hardware engine.

The operating system used is TinyOS-2.1.1 (P. Levis, 2006), used in many other works as well. It is based on the nesC language (Gay et al., 2003), developed for platform flexibility, and cross-platform networking.

6.3.2 Experimental Procedure

The private keys were installed in the program code. They were not exchanged by radio but the exchange simulated by storing the public key *ID* of the neighbour as a variable. The code was kept to the bare minimum for key computation and switching on the LEDs at the start and end of 100 computation iterations. The time taken was measured using a stop watch. The power supply to the node was regulated at 3.1 V and the average current during computation was measured to be 8.7 mA. This is used for estimating the energy used. The RAM and ROM requirements were obtained from the TinyOS compiler outputs.

6.3.3 Performance Measures

ROM Requirements The main storage requirements are for the private key-sets and the programme code. The private key-set requires a storage of $Q_o = \eta Nm \times b$ bits. This is static and can be stored in flash memory. The number of bits b used is ≤ 5 bits. To simplify coding, 1 byte is used for each data unit. This results in a wastage of 37.5%

Listing 2: BYka Pairwise Key Computation

Input: Neighbour Node's public key-tag s_B
Output: The pairwise key K_{pair}
Initialise $R_{A_{ik}} = 1$ # Initialise and make non-zero
for $j = 1$ to η **do**
  $V_{B_j} = (s_B + (j-1))^{u-1} \pmod q$
  for $i = 1$ to η **do**
    for $k = 1$ to N **do**
      for $u = 1$ to m **do**
        $R_{A_{ik}} = R_{A_{ik}} + K_{A_{ik_u}} V_{Bj} \pmod p$
      end
    end
  end
end
$K_{pair} = 1$
for $i = 1$ to η **do**
  for $k = 1$ to N **do**
    $K_{pair} = K_{pair} \times R_{A_{ik}} \pmod{K_s}$
  end
end

if $b = 5$ bits and 50% if $b = 4$ bits. If necessary, to save memory, the code can be written to splice up the 8 bit memory spaces and fully utilise all the bits for storing the private keys. This would require additional lines of code and was not investigated in this study. The storage for the private key-set is then $Q_o = N\eta m$ bytes.

BYka Key Computation Code Consider that node A has obtained node B's *ID* s_B. In our implementation, the u^{th} public key vector element $V_{B_{j_u}} = (s_B - \overline{j-1})^{u-1} \pmod q$ is generated once and used with all the u^{th} private key elements to obtain $R_{A_{ik_u}} = \sum_{u=1}^{m} K_{A_{ik_u}} \cdot \left[(s_B + \overline{j-1})^{u-1} \pmod q\right] \pmod p$. This requires only one byte in RAM instead of generating and storing all the ηm elements in $V_{B_{j_u}}$. The computation pseudo code is given in Listing (2).

RAM Requirements During execution, RAM is required for some counters, the public vector, the key-set R_A of $N\eta^2$ integers, and the final pairwise key of $log_2(K_s)$

bits, where K_s is a prime of the desired key size. While all the $m\eta$ elements of the public keys need to be computed, the code is written such that only one element is computed and used at a time, using only one memory space in RAM.

Overall, the largest amount of RAM required is for the pairwise key-set, $Q_R = N\eta^2 \times b$ bits. If one byte is used for each b bits element in R, then the RAM required is,

$$Q_R \quad = \quad N\eta^2 \quad \text{-bytes}$$

The requirement for RAM is very small. Even with large values of $N = 12, \eta = 4$, $Q_R = 192$ bytes plus 1 byte for each of the counters i, j, k and 2 bytes for the seed s_B.

Pairwise Key-set Computation Time The pairwise key-set computation involves $N\eta^2$ rounds of matrix multiplications, each one involving two steps: generating the public key vector, and multiply it with the pairwise key to obtain a pairwise key-set element. After these, the final pairwise key is computed from the pairwise-set integers either by sorting and concatenation, multiplication modulo K_s, or counting the occurrences and input into a hash function. We used sorting and concatenation in our case.

The public key generation requires $\eta(m-2)$ modulo q multiplications since the first two terms are obvious. The pairwise key-set involves $\eta \times N\eta \times m$ modulo p multiplications and $\eta \times N\eta \times (m-1)$ modulo p additions. The sorting into bins involves $N\eta^2$ operations. Overall the computation time can be given by,

$$T_{comp} \propto [\eta(m-2) + N\eta^2 m]_{mult} + [N\eta^2(m-1)]_{add} + [N\eta^2]_{sort}$$

With data size of b bits, the complexity for multiplication is $\mathcal{O}(b^2)$, compared to $\mathcal{O}(b)$ for additions and sorting. Since the multiplication operations are considerably slower than additions or sorting, the latter two are ignored and the computation time

η	N	$m = 12$		$m = 16$		$m = 24$	
		ROM	Time.	ROM	Time	ROM	Time
6	7	504	156	672	200	1008	288
	8	576	174	768	225	1152	325
7	6	504	178	672	229	1008	332
	7	588	203	784	263	1176	383
	8	672	228	896	296	1344	433
8	6	576	224	768	292	1152	426
	7	672	257	896	335	1344	491
	8	768	290	1024	379	1536	557

Table 6.4: BYka Key Computation Times (ms) and Private Key ROM storage (bytes) for Various Parameters of m, N, and η

can be approximated to,

$$T_{comp} \propto [mN\eta^2 + (m-2)\eta]$$

The computation times and ROM requirements are given in Table 6.4. The linearised experimental results for computation times are plotted in Fig. 6.3. From this graph, the computation time is,

$$T_{comp} = 0.0428[mN\eta^2 + (m-2)\eta] + 23.72 \quad \text{milliseconds} \tag{6.2}$$

Energy Consumption The energy consumed in key agreement schemes comprises those required for initial exchange of credentials, verification of the credentials, and the actual pairwise key computations. In our scheme, the number of computation operations is deterministic and we can assume that the energy consumed for key computation is proportional to the computation time. From the experimental results on the MICAz mote, it was found that the average current drawn during the computation from the 3.1

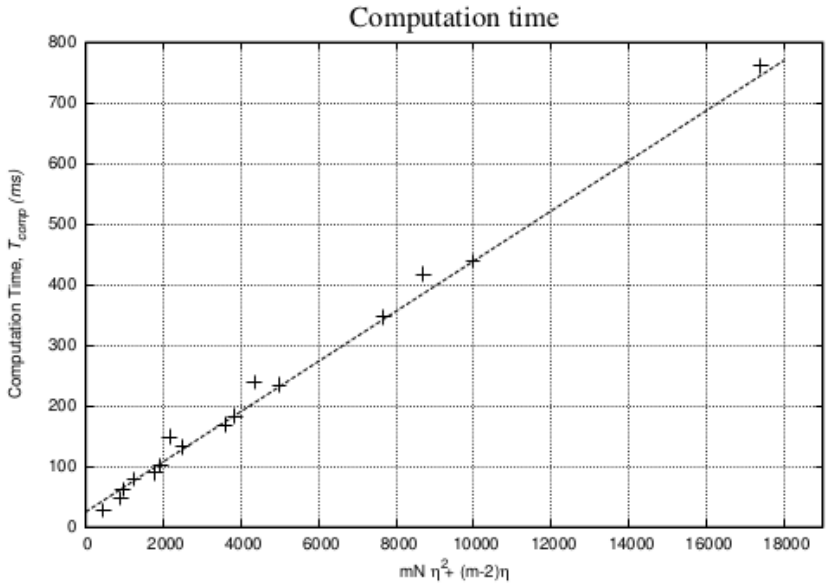


Figure 6.3: Graph of Pairwise Key Computation Times Using the MICAz mote

V regulated power supply was measured to be about 8.7 mA. Using this, the estimated energy in mJ, used for computation was estimated as $(3.1 \times 0.0087 \times T_c)\ mJ$.

Communication Overhead for Public Key Exchange In a key agreement protocol, a pair of nodes must exchange some information such as their public keys to commence the protocol.

In the BYka scheme, the nodes need to exchange their public key *ID* which is only 16 bits in length. The scheme does not have a separate credential verification step. It merely assumes that the provided *ID* is correct and verification is implicit in the success of obtaining an identical pairwise key. Since the pairwise key computation is very fast, this is acceptable. In schemes where the pairwise key computation time is significantly larger than the verification time, the key verification must first be satisfied.

In the IEEE 802.15.4 wireless protocol commonly used for sensor networks, a single frame capable of 2 to 127 bytes of payload data (IEEE Computer Society, 2006), would be sufficient to transmit this public key *ID*.

Overall, the resource requirements and computation time is reduced by using smaller values of the three keying parameters N, η and m.

6.4 Summary

The BYka scheme was implemented on computer and capturing of the nodes was simulated by storing the nodes into a pool as they were created. As a new node is "captured", it is paired with each of the nodes in the pool, and using the pairing attack, the computed pairwise key-set examined to find a traitor node. This was repeated with new nodes created and added to the pool until finally a traitor node is found. The execution times can be very long and suitable keying parameters were chosen to yield results in a period of days. The number of nodes required to capture in order to find a traitor node, n_c is about $6\% \sim 12\%$ smaller than the estimated values for keying parameters used. The estimated number of possible master key solutions Φ, which is considered as the security strength, is consistently smaller than the experimental values. This may be due to the approximations of the distributions, but overall, shows that the theoretical estimates are conservative.

The BYka scheme was implemented in the MICAz sensor nodes and the key computation times for various keying parameters were obtained. This enables the key computation times in the MICAz motes to be estimated for various configurations of keying parameters.

Chapter 7

Implementation and Application

7.1 Introduction

The BYka scheme has been shown to be secure and resilient against an adversary who is able to capture any number of nodes and has powerful computing resources. This chapter discusses how the keying parameters for the scheme can be selected in practice. The configuration of the keying parameters can be selected for the desired security strength, memory availability, or computation times. A set of parameters giving the indicative fastest key computation times when implemented in the MICAz mote using the code in Appendix B.4 is suggested. The BYKa scheme can be used as the cryptographic primitive in a variety of scenarios. As an example, an authenticated message protocol using the BYka scheme is proposed for use in very dynamic mobile ad hoc applications.

7.2 System Implementation

7.2.1 Design Equations

The equations required to select the appropriate keying parameters are gathered here for easy reference. They are grouped into three areas: (1) security requirement, (2) resilience against node capture, and (3) performance.

1. Security Requirement

The pairwise keyspace size is the most stringent security requirement, which if met, will make the keyspaces for all the other keys sufficiently large to resist brute force attacks.

The pairwise keyspace, from Eqn.(3.9) is,

$$K_{sp} = \binom{N\eta^2 + p - 1}{p - 1} \tag{3.9}$$

2. Resilience Against Node Capture

Traitor node If a traitor node can be found, the PPMka of the private keys in the node are exposed. This does not break the scheme but increases the chance of exposing the PPMka in other nodes. The probability of finding a traitor node, from Eq. (5.6) is,

$$\left.\begin{aligned} P_t &= \frac{2\theta_u - \theta_d}{p^{2N\eta - N}} \\ &\text{where,} \\ \theta_u &= \sum_{r=1}^{N_a} \binom{p}{r} Q_{N_a r} (p-r)^{N\eta} \\ \theta_d &= \sum_{r=1}^{N_c} \left[\binom{p}{r} Q_{N_c r} \times \sum_{k=1}^{N_a} \binom{p-r}{k} Q_{N_a k} (p-r-k)^{N_b} \right] \\ Q_{N_a r} &= r^{N_a} - \sum_{j=1}^{r-1} \binom{r}{j} Q_{N_a j} \;\text{ and }\; Q_{N_c r} = r^{N_c} - \sum_{i=1}^{r-1} \binom{r}{i} Q_{N_c i} \end{aligned}\right\} \tag{5.6}$$

Traitor Node Capture Size The number of nodes expected to be captured to find a traitor node from Eqn. (5.7) is,

$$n_c \geqslant \frac{1}{2}\left(1+\sqrt{1+\frac{8}{\eta^2 P_t}}\right) \tag{5.7}$$

Security Strength The number of trials Φ required to find the master keys, which is a measure of the number of steps required to break the scheme, from Eqn. (5.15),(5.16), and (5.17) is,

$$\left.\begin{aligned} &\Phi = \sum_{i=0}^{N-1}[\mu - i]^m \\ &\text{where} \quad \mu = P_t N\eta \\ &\text{and} \quad P_t = \binom{N\eta}{N} p_r^N (1-p_r)^{(N\eta - N)} \end{aligned}\right\} \quad (5.15), (5.16), (5.17)$$

3. Performance Requirements

Memory The approximate ROM and RAM required are given by,

$$Q_o = N\eta m \text{ -bytes} \tag{7.2a}$$

$$Q_R = N\eta^2 \text{ -bytes} \tag{7.2b}$$

Computation Time The expression for the pairwise key computation time in the MICAz mote using the TinyOS code in Appendix B.4, from Eqn. (6.2) is given by,

$$T_{comp} = 0.0428[mN\eta^2 + (m-2)\eta] + 23.72 \quad \textit{ms} \tag{7.2c}$$

Network Size This is the number of unique *ID*s available using the prime q, i.e.,

$$\Omega \approx \frac{q}{\eta} \tag{7.2d}$$

7.2.2 Selection of Parameters

The value of m does not affect the traitor node capture size n_c or the pairwise key size. It does have a strong impact on the number of permutations Φ required to compute the master keys as given in Eqn. (5.17). However, as we see in Table (5.4), for the values of N, η used, the values of Φ are all extremely large so that using $m = 24$ is suitable for most situations. The computation time is most affected by the master key matrix size m, the number of keys N, and the number of public keys η. The keying parameters can be selected based on minimum ROM storage, or computation time for the desired security strength and node capture size. The Tables given in Appendix A, computed for various parameters, can be used as a guide to implement the desired design.

Optimum Parameters

An exhaustive search of the configurations to obtain the fastest key computation times for various security strengths of Φ, while satisfying the minimum traitor node capture sizes of 10,000 nodes or more is presented in Table (7.1). The values are based on the MICAz mode using the TinyOS code given in Appendix B.4. They are indicative only, and there may be further gains by optimising the code for speed and efficiency. The storage requirements for the private key range from 468 bytes for 64 bits security strength to 1824 bytes for 192 bits security strength. The computation times are fast. The 64 bits security strength requires about 85 milliseconds, while the longest computation time was 342 milliseconds for 192 bits security strength.

Security Strength	n_c	Φ	$Q_o(B)$	$T_{comp}(ms)$	p	m	η	N
192	6.63×10^4	1.00×10^{58}	1824	342	61	38	4	12
128	1.38×10^7	9.04×10^{38}	1170	279	31	26	5	9
112	4.55×10^5	2.33×10^{36}	920	185	31	23	4	11
80	2.30×10^4	1.84×10^{25}	612	104	31	17	3	12
64	1.02×10^6	8.47×10^{18}	468	85	17	13	3	12

Table 7.1: Optimal Parameter Based on Related Security Strength, T_{comp}, and Traitor Capture Size n_c. Φ is the Number of Possible Trials Required.

		Number of Years				
Processor	flops	Φ_{192}	Φ_{128}	Φ_{112}	Φ_{80}	Φ_{64}
Sequoia	5.15×10^{23}	1.94×10^{34}	2.68×10^{14}	4.53×10^{12}	3.58×10^{1}	0

Table 7.2: Number of Years to Break the BYka Scheme Assuming one flop per Trial

Computing Resources Required to Break the Scheme

The number of trials Φ required to obtain the master keys involves constructing a $(m\times m)$ system of equations and then solving them for the master key, testing it to see if it can be used to successfully compute the private key, and repeating the process until all the master keys are found. Each solution is of $\mathcal{O}(m^3)$. As an indication of the time required, consider the Sequoia supercomputer which is able to execute 16.32 petaflops (Advanced Simulation and Computing, 2012) or 5.15×10^{23} flops per year. If we consider the grossly oversimplified situation where each trial requires a single flop in the supercomputer, the number of years it will require to break the scheme can be up to 5.15×10^{23} years for the 192 bits security strength case, as given in Table (7.2).

7.2.3 Key Generation and Distribution

The BYka scheme requires that all the public and private keys used are obtained from a central body, the Trusted Authority (TA). The TA can be a single entity for small systems, or a group of peer entities for large installations. The key distribution operations can also be done centrally or distributed among subsidiary agents. Interestingly, the TA can also operate as a "central committee" (CC) with each member being independently responsible for a subset of the master keys which are used to generate sub-sets of the nodes' private key-sets. The public key *ID*s may be assigned by one of the CC members, another separate entity, or by each CC member from a subset of *ID*s, as long as they are unique. As each node is created, it is passed from one CC member to another to obtain its contribution of the private key-set.

The overall task of generating and providing each node can thus be distributed among the CC members but each member must individually play its part in the process.

7.3 Applications

The BYka scheme allows member nodes to obtain pairwise keys with each other very quickly and efficiently. It is deterministic and there is no need for the participation of a third party. The initial exchange is the *ID* which is only a few bits. There is no need to protect the *ID*. These features allow the BYka scheme to be used in very dynamic mobile ad hoc situations. The following describes some applications where the BYka scheme would be useful as the cryptographic primitive.

7.3.1 The Single Message Authentication Protocol (SMAP)

In a highly dynamic ad hoc mobile environment, for example in surveillance, wildlife monitoring, or vehicular networks applications, if a mobile node comes into range with

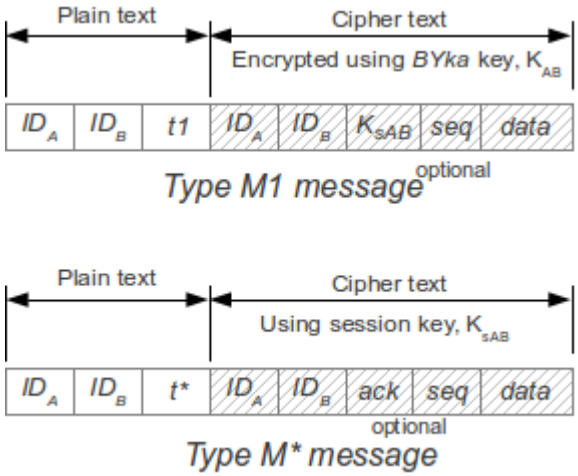


Figure 7.1: Message Format from Node A to Node B

another member node, it may need to send its data quickly before it moves out of range. The data may be in single or multiple messages but they must be authenticated before they are processed or relayed. A response may, or may not be required, and there may also be an exchange of several messages. The environment may be fragile and messages can be lost, corrupted, or the nodes simply moved out of range.

For this scenario, the Single Message Authenticated Protocol (SMAP) is proposed, described as follows.

Message Format

All messages between pairs of nodes consist of a plain text and a cipher text part similar to the format in Fig. (7.1). Both parts include the source and destination *ID*s. The cipher text part can also carry a confidential payload as well.

Message Types

The protocol defines two types of messages, type *M1* and type *M**. A type *M1* message is used when the sender has not established a session key with the receiver, for example in the first message from the sender to the receiver. It is identified by the tag $t1$ in the

plain text part. The cipher text part is encrypted using the "BYka key" K_{AB}, computed using the BYka scheme.

The second type of message *M**, identified by the tag $t*$ in plain text, is used to send messages to a node from which it has previously received a message. This type of message is normally used for multiple exchanges. The cipher text part is encrypted using the session key, K_{sAB}, a randomly generated key.

Node *ID*s

Nodes would learn about other nodes in the neighbourhood by monitoring the plain text part of messages. In addition, a newly deployed node, would broadcast an advertisement which is just a plain text message of type *M1* containing its *ID*. If a pair of nodes do not share a session key, the node which has data to send becomes the initiator and the other, the responder. Here, we typically refer to nodes A and B as initiator and responder respectively.

SMAP-1 Mode

Communications between pairs of nodes start in this mode – the Single Message Authentication Protocol-1 (SMAP-1). The main emphasis here is to send a message securely and quickly.

Initiator Node A Consider that node A has some data to send to node B. It has obtained ID_B by monitoring messages. It computes the BYka key K_{AB} using ID_B, and sends the message to B using message type *M1*:

$$(M1)\ A \rightarrow B : \langle ID_B, ID_A, t1, E_{K_{AB}}(ID_B, ID_A, K_{sAB}, data)\rangle$$

The cipher text part containing the *ID*s, a randomly generated "proposed session key" K_{sAB}, and data, is encrypted using the BYka key, K_{AB}. If an acknowledgement is required, a sequence number is included in the cipher text. The BYka key and proposed session key are stored for a time T_s in case they are needed again.

At this point, node A believes that only a node with ID_B belonging to the same TA would be able to decrypt and verify the cipher text in the message.

Session Key Acceptance The proposed session key K_{sAB} has to be "accepted" before it can be used. The responder node B accepts the key after validating the *ID*s in the cipher text in *M1* message. On the other hand, the initiator node A will only accept its own proposed session key after receiving a valid type *M** message from the responder, i.e. after it knows that the proposed key has been received and accepted, see Figs. (7.2) and (7.3).

Responder Node B When node B receives the message, it recognises the message as a type *M1* message from the plain text tag $t1$. It obtains ID_A, computes the BYka key, and decrypts the cipher text. If the encrypted *ID*s and the plain text *ID*s match, the sender of the message ID_A is authenticated.

At this point, node B believes that it shares the session key K_{sAB} with a node with identity ID_A, and that they both belong to the same TA.

Node B accepts the proposed session key K_{sAB} and, as with the BYka key, it is stored for time a length of time T_s. If no acknowledgement is required, the protocol completes.

If required, a response containing an acknowledgement is sent in a message of type *M**, encrypted using the accepted session key K_{sAB}.

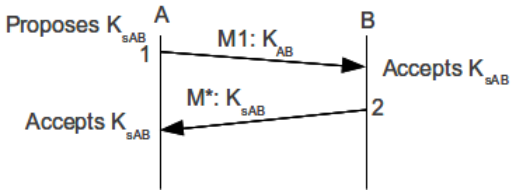


Figure 7.2: SMAP-2

SMAP-2 Mode

In situations where there are many messages, a session key would be preferred as it reduces the exposure of the BYka keys. After node B has received a type *M1* message from node A, it establishes a session key for sending data back to A. Any subsequent message to node A will be of type *M** encrypted using the session key K_{sAB}. This is the SMAP-2 mode.

If node A receives a response message of type *M** from node B, e.g. an acknowledgement, before the timer T_s expires, node A would also accept the proposed session key K_{sAB} which it had previously stored. The session key becomes established and a secured link now exists between them. Node A now switches to SMAP-2 mode for future communications with node B until the timer T_s expires.

An ideal run of the protocol is shown in Fig. (7.2). Here, only the message types and the encryption keys used are shown for clarity, i.e. *M1* encrypted using the BYka key K_{AB}, and *M** encrypted using the session key K_{sAB}.

Overall Operations

The protocol starts in SMAP-1 mode and opportunistically switches to SMAP-2 mode if a response message is received. Otherwise it continues using the SMAP-1 mode.

The protocol SMAP-2 may look like a two message protocol. However, when node B receives the first message *M1* in *run 1* in Fig. (7.2), it can immediately authenticate node A and start using the session key for future messages to node A. If node A

does not receive the reply in *run 2*, the next message from node A to node B would continue using type *M1* message encrypted using the BYka key, effectively restarting the SMAP-1 protocol.

Also, once a node receives a type *M** message from its counterpart, it switches to the mode 2 SMAP-2 protocol. The BYka key can then be deleted for added security. This happens in node A after it receives a response to its initial type *M1* messages, but only happens in node B after the second or subsequent message from node A.

The SMAP-2 mode can be used in an unreliable or reliable communication method. If lost messages can be tolerated, the unreliable method would be used. The reliable mode requires the use of sequence and acknowledgement numbers and a response timer T_r such that if a message is not acknowledged within this time, it is retransmitted.

Reliable Method

All messages would include a random sequence number and if applicable, an acknowledgement number related to the previous message received. The sequence numbers and acknowledgement numbers are encrypted in the cipher text part.

The initiator node after sending the first message of type *M1*, sets a response timer T_r. If a reply is not received in time, possibly due to the message or the response being lost or corrupted, the same message is retransmitted, see Fig. (7.3).

After node B receives and validates the message, it constructs a response containing the acknowledgement *ack*, a new sequence number *seq* and any data, encrypts it using the session key K_{sAB} and sends it as a type *M** message.

Unreliable Method

The protocol operates in the same way as the reliable method but no sequence numbers and acknowledgements are used.

While the reliable method will always switch to SMAP-2 mode, the unreliable method will opportunistically switch from SMAP-1 to SMAP-2 mode if the initiator subsequently receives a message of type *M** from the responder before the time T_s expires.

Robustness

Lost Messages Messages can be lost or become corrupted but this does not affect the protocol. If the initiator's message is lost, both nodes can fall back to using SMAP-1 mode. Also, the initiator would switch to SMAP-2 if it receives a response, otherwise it will continue to send messages to the responder using type *M1*, effectively restarting the protocol.

Incomplete Runs The session key is only used by a node after it has been accepted. If a run in the protocol is incomplete, the node which has not accepted the session key would not be able to use it. Instead it would send any subsequent message using type *M1*, effectively falling back to SMAP-1. However the node which has accepted the session key would be able to use it in subsequent messages of type *M**. In this way incomplete runs due to interference and nodes going out of range do not affect the protocol as it is able to fall back to SMAP-1 mode. The following considers other impacts of incomplete runs.

Consider Fig. (7.3). If *run 1* from node A to node B is incomplete and an acknowledgement is required, the sender retransmits the same message in *run 1a* after the response timer expires. If the response in *run 2a* is lost, node A would, in *run 1b*, retransmit the message after the timer expires. If node A does not receive any response from node B after retransmitting for a certain number of times, e.g. *run 2a* and *run 2b*, it gives up. Node A has not accepted the session key with node B and the next message to B would use message type *M1*. However, if a message, possibly unrelated to earlier

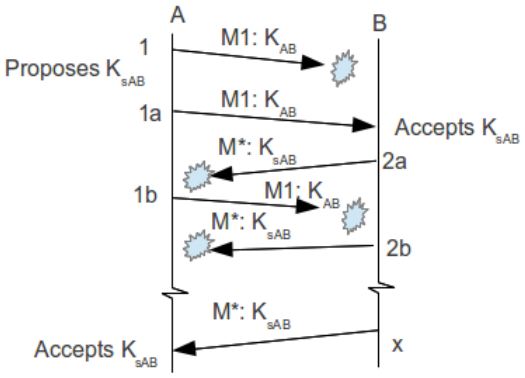


Figure 7.3: SMAP-2 with Incomplete Runs

messages, is received from node B (of type *M**) in *run x* before the time T_s expires, node A would take the opportunity to accept the session key thus completing the key establishment.

Attacks

Confidentiality of messages All the sensitive information is encrypted using keys derived from the endpoint *ID*s and private keys. An attacker without a valid *ID* and the related private keys would not be able to compute the BYka key to read the cipher text.

MITM Attack In the man-in-the-middle (MITM) attack, the attacker is unable to read the cipher text and cannot establish a secure link with any node as it does not have the private key-set to compute the pairwise BYka key. The man-in-the-middle (MITM) attack only results in the attacker helping to forward messages.

Protocol Manipulation The protocol has at most two runs. There is not much that can be done to manipulate the protocol runs. In the SMAP-1 mode, there is nothing to manipulate except to send fictitious or replay messages. This can cause the target node to expend some energy to compute futile BYka keys.

The attacker can send fictitious messages as well as replay messages with modifications but these would be discarded as they would fail to be validated. Fictitious messages of type *M1* will cause the target to expand resources to compute the BYka keys and decrypt the cipher text. Fictitious messages of type *M** will be more benign as they only require the stored session key for decryption.

The number of bits to exchange for authenticating each other's *ID*s and contents is very small. Apart from payload data, the message of type *M1* needs to include 4×2 bytes *ID*s, and the 16 bytes session key, a total of 24 bytes.

7.3.2 Corporate Email

It may be desirable for emails between members in corporations to be encrypted, and yet readable by senior management. Initially on joining, a new staff member is given an *ID* and the related private key, possibly installed in the computer provided. The new staff's name, email address, and *ID* may be published in a directory system. A mechanism can be built into the email client software to automatically look up the recipient's *ID*, compute the pairwise key and encrypt the message before sending it to the mail server. The receiver's mail client, using a similar mechanism obtains the sender's *ID* from the message header, computes the pairwise key and decrypts the message. The resource requirements is minimal and allows it to be used with all kinds of devices including tablets and phones.

As the pairwise key computation is non-interactive, messages can be sent to potential, new staff.

7.4 Comparison with other Schemes

A comparison of authenticated establishment schemes is shown in Table (7.3). The schemes using PKC algorithms are not comparable to symmetric key schemes in terms

of computation times and memory requirements for the same security strength. In fact, the reported results in the literature are up to only 80 bit security strength. The scheme based on IBC which, like the BYka scheme, does not need a separate mechanism for entity authentication is much faster, achieving 80 bit security strength in a time of about 1.9 seconds. Symmetric key schemes based on Blom or Blundo are much faster. The computation times should be similar as the number of computations to obtain the same pairwise key size are the same. While the scheme in (W. Zhang et al., 2007) can obtain keys of 80 bits in about 130 milliseconds, our BYka scheme can achieve this in 104 milliseconds. In addition the BYka scheme can compute 128 bit keys in 279 milliseconds, which seems slow compared to 140 milliseconds in (Yu et al., 2010) which used 16-bit μC compared to our 8-bit device.

7.5 Summary

The BYka scheme can be implemented for various requirements of security strength, storage memory, and computation time by careful selection of the keying parameters of m, N, η, p and q, using the equations gathered from the previous chapters. The parameters based on the fastest computation times of less than 342 milliseconds in the MICAz mote is given for security strengths of $\{64, 80, 112, 128, 192\}$ bits. All the keys including the master keys can be managed by a single entity, or for added security, by a group of independent entities. This also allows the task of bootstrapping the nodes to be distributed among several entities.

The application of the BYka scheme as the cryptographic primitive for the single-message-authentication-protocol (SMAP) demonstrates its utility for use in very dynamic mobile ad hoc networks. Even though the BYka scheme is suitable for low resourced devices, it can also be used in other applications such as protecting the email communications within an organisation.

	PKC		IBC	Symmetric key		
Scheme	TinyPK	TinyECC	Tiny PBC	Zhang'07	Yu'09	BYka
Year	2004	2008	2011	2007	2010	2014
Processor	†ATmega-128	Atmega-128	ATmega-128	ATmega-128	‡MSP430	ATmega-128
Primitive	RSA-1024	ECDH-ECDSA	Tate-pairing ECC	Blundo	Blom	Blom
ROM	12.4 KB	19.3 KB	37.9 KB	15 KB	8.9 KB	7.79 KB *incl. 1.536 KB prv key
RAM	1.17 KB	1.5 KB	3.6 KB		0.33 KB	0.61 KB
Comp. time	14.5 s public key operation.	*6.2 s after 5.2 s initialisation	1.9 s	130 ms	140 ms	$104 \sim 279$ ms
Exchange bits	2048, estimated	2200 bits, est.		128 bits		16 bits
Key size (bits)	80	80	80	80	128	$80 \sim 128$
Comments	Priv key: 10's of seconds	*All optimisation on. 120 s w/o signatures				
Source	(Watro et al., 2004)	(A. Liu & Ning, 2008)	(L. B. Oliveira et al., 2011)	(W. Zhang et al., 2007)	(Yu et al., 2010)	This Thesis

Table 7.3: Comparison of Authenticated Key Agreement Schemes for Sensor nodes, † 8-bit μC @ 8 MHz, ‡16-bit μC @ 8 MHz.

Some comparison with other authenticated key agreement schemes showed that the computation times in the BYka scheme are more than $10\times$ faster than the fastest IBC schemes based PKC algorithms, while comparable if not faster than other schemes using symmetric key algorithms. The number of bits to exchange is extremely small, only the public key *ID* of 16 bits. Compared to PKC schemes which have to exchange public keys of hundreds of bits, (at least 320 bits for ECC algorithms, excluding certificates), this is a huge saving in energy for communications. The memory requirement for the BYka scheme does not increase as the network size increases. At a maximum of 1824 bytes for a security strength of 192 bits, it can be used in networks that can allow 66,000 nodes to be captured.

Chapter 8

Conclusion

8.1 Introduction

The full potential for wireless sensor networks can be realised, especially for security sensitive applications, if the receiver is able to verify that the received messages come from an authentic source, the contents have not been altered by an adversary, and if necessary, are encrypted for confidentiality. Cryptographic tools can be used to protect the messages for confidentiality, integrity, and authenticity, and are already widely used in computer networks. These require that pairs of communicating nodes share a common secret pairwise key.

8.2 Key Agreement Schemes

Authenticated key agreement schemes are desirable for providing the pairwise keys in mobile ad hoc networks as the nodes can compute their own keys when required, with the assurance that both parties belong to the same organisation. Many such schemes are already widely used in computer networks. Adapting them for sensor networks is difficult as these use public key cryptographic algorithms which require substantial

computing power, memory and energy resources. On the other hand, symmetric key cryptographic methods which do not require much resources are more suitable for the low resourced sensor nodes.

An interesting symmetric key scheme is the Blom's key agreement scheme which has useful attributes for sensor networks. It is mutually authenticating as all the nodes must obtain their keying material from the trusted authority (TA) to successfully compute their common pairwise keys. However, if an adversary is able to capture a certain number of nodes, called the capture threshold, the master key can be derived, breaking the entire scheme. A large capture threshold requires proportionally large amount of keying material to be stored in the nodes making it impractical for large scale use. Many attempts have been made to improve its resilience against node capture without the proportional increase in storage requirements such as by making it more difficult to obtain the required keying material using multiple key spaces, or by obfuscating the links between the private keys and the master key by introducing some random perturbations.

8.3 Research Objective

This study set out to study whether it was possible to modify the Blom's symmetric key agreement scheme using permutations of multiple keys so that it is secure and practical for use in large sensor networks where any number of nodes can be compromised. This is shown to be possible by developing the Blom-Yang key agreement (BYka) scheme which can be tailored for different situations. For example, to accommodate 23,000 nodes on a network using the MICAz motes, the implementation using the TinyOS code given in Appendix B.4 requires a storage of only 1824 bytes for the keying material, achieving a security strength of 192 bits, with a key computation time of 342 milliseconds.

8.4 The BYka Scheme

In the proposed BYka scheme, the TA has multiple master keys and each node is assigned multiple unique public keys. The TA uses them in permutations to compute multiple private keys for each node and stores them in a random order in the node. Now, if the node's private keys are stolen, they cannot be used without knowing which public key and master key was used to compute the private key. This private-public-master-key association (PPMka) information is not available anywhere and there is only a very small chance of getting all the correct PPMka in order to mount attacks on the scheme. Since the private keys are random integers stored in a random order, the brute force attacker has to try an infeasibly large number of possibilities to find the correct PPMka.

The attacker can attempt to find the PPMka by getting a pair of captured nodes to compute their pairwise keys using each other's public keys. The sets of integers, called the pairwise key-sets, forming the pairwise key obtained in each node are identical. If all the integers are unique, the identical numbers across both sets link the related private keys to the associated public keys and master keys, exposing the PPMka. However, if the numbers are not all unique, then there are ambiguities. In the BYka scheme, the key computations are defined over a very small prime field $\mathbb{F}_p$, for example $p = 31$. This ensures that the key-set integers are not all unique, thereby preventing the PPMka from being discovered easily.

Security Strength

The most efficient attack to discover the PPMka is the pairing attack where a pair of nodes uses only one of each other's public keys to compute a partial set of the pairwise key-set. The size of the partial key-set is much smaller, increasing the chance of the integers being all unique. Even then, by selecting suitable values; the number of master keys, the number of private keys, the master key size, and the size of the prime modulus,

the probability of finding a node which exposes its PPMka, called a traitor node, can be made extremely small. From the detailed study of this attack, analytical results were obtained to estimate the number of nodes required to be captured to find a traitor node, n_c. With suitable parameters, the value of n_c can be in the tens of thousands of nodes. If the network size is less than n_c, then a traitor node virtually cannot be found. The adversary can also try all the possible solutions of the master keys from the possibilities available in each pairing attack. The analytical results enable the number of possibilities to be estimated. This showed that the number of iterations required can be 2^{80}, 2^{112}, 2^{128} or even 2^{192}, making the security strength of the scheme at least $80, 112, 128$ or 192 bits, respectively.

The analytical results were verified against those obtained from computer simulated attacks on some implementations. In addition, implementations of the BYka scheme were done using the MICAz mote to show its practicality, and to obtain some data on the key computation times. The outcomes were used to set out the guidelines and tables for the practitioner to select suitable keying parameters for the desired computation times, memory requirements, pairwise key sizes, and resilience against node capture.

Performance Features

The scheme is fast, requires only a few lines of code involving modulo multiplications and additions, and the storage requirement does not grow proportionally with the network size. For example, to cater for networks of up to 23,000 nodes, the key computation times range from 85 milliseconds to 342 milliseconds for security strengths of 64 to 192 bits respectively. The corresponding keying material size is 468 to 1824 bytes. To initiate the pairwise key computation, each node needs to transmit their public key *ID* which is a 16 -bit integer. This small amount of exchange saves energy used for radio transmission. If the *ID* of the receiver node is already known, the sender is able to, without any interaction between them, immediately compute their pairwise key

and use it to encrypt a message for the receiver. This ability would be useful in highly mobile ad hoc networks. An application using this scheme was proposed, the single message authenticated protocol (SMAP).

The BYka scheme operates in an exclusive posture in that it only allows nodes belonging to the same TA to establish pairwise keys with each other. Outsiders are excluded. All member nodes share a common heritage which is their private key-set derived from the same set of master keys in the TA. This common heritage enables a pair of nodes to implicitly authenticate each other if they are able to compute a common pairwise key. Operationally, it is effectively an identity-based scheme though, unlike identity-based schemes based on bilinear pairings where the *ID* can be any string, here the *ID* is an integer. It is a non-interactive scheme and the sender, having obtained the receiver's *ID*, is able to compute their pairwise key and encrypt messages for it even without the receiver being present. This opens up the scheme to other applications where encrypted messages may be sent to nodes which will be encountered in the future, such as for email to an employee who is about to join the company.

The TA is a key escrow entity and must be kept secure. It is able to compute the pairwise keys of all the nodes and with it, decrypt all previously recorded messages. While this appears to breach privacy concerns, it may actually be a desirable feature in some situations such as in business corporations. To help with distribution, and possibly for additional security, the TA's role may be distributed among a committee of TAs who must jointly work together. If necessary, the key escrow privileges may be relinquished by deleting all the master keys after generating all the possible public and private keys that will ever be needed.

Key agreement schemes are generally vulnerable to the man-in-the-middle (MITM) attacks, especially where the nodes generate their own public and private keys. Additional measures are usually required for authenticating the public keys. However, in the BYka scheme, the public and private keys are not self generated but by the trusted authority

and it is not possible for the MITM attacker to compute the pairwise key with a legitimate node without the required private keys. This makes the BYka scheme immune to the MITM attack.

8.5 Future Work

8.5.1 Identity Theft

One of the vulnerabilities of the BYka scheme is the identity theft attack where the adversary, having obtained the keys of a compromised node, uses them to make rogue nodes with the same keys. In addition, a rogue node would be able to mount the compromised-key impersonation attack where it is able to impersonate any node to the compromised node. Identity theft is a serious threat but is not within the scope of this study. At this time, it is assumed that a compromised node can be detected and its *ID* distributed for excommunication. Further work is required to study how to identify compromised nodes and to develop suitable countermeasures.

8.5.2 Forward Secrecy

The BYka scheme does not provide forward secrecy. If an adversary obtains a node's private keys and has access to all previously recorded messages including those transporting the randomly generated session keys, then the adversary would be able to decrypt all the previous messages.

8.5.3 Non Linearly Independent Public Keys

The Blom's scheme requires the use of linearly independent public keys to prevent the Sybil attack using keys stolen from the captured nodes. In addition, the master key

can be derived from any m captured private keys. This limits the number of nodes to $(m-1)$ if the system is to be unconditionally secure.

In the BYka scheme, the unknown PPMka of captured private keys leads to an interesting proposition for future study: If the PPMka is unknown, the Sybil attack cannot be mounted. This would remove the need for the public keys to be linearly independent and implementations using linearly dependent public keys may be considered. Since linearly dependent public keys give rise to the associated private keys being linearly dependent as well, then the system of equations constructed from the m captured private keys do not have determinate solutions. Consequently, the master keys can never be found. The question then is how much more resilient the system would be in addition to the difficulty of discovering the PPMka. There is also the potential to use public key vectors which are arithmetic progressions making the computations even easier and faster.

8.5.4 SMAP

The non-interactive nature of the BYka scheme enables a node to encrypt messages for another node just from knowing the *ID* of the receiver. This enables nodes to send protected messages to destination nodes whenever opportunities arises, for example, in highly mobile networks where nodes are not normally within range of each other. This is implemented in the SMAP protocol. Further work will be done to test its operations and performance.

8.6 Summary

This thesis demonstrated that an authenticated key agreement scheme based on a symmetric key algorithm, the proposed BYka scheme, is secure for highly dynamic,

mobile and ad hoc wireless sensor networks and does not require increasing storage for the private keys as the network size increases. Nodes are first authenticated by the TA and then provided with their keying material. After deployment, pairs of nodes only need to exchange their *ID*s consisting of a few bits in order to compute their common pairwise keys. The BYka scheme is mutually authenticating and immune to the man-in-the middle attack. It can be configured for the required security strengths of up to 192 bits against a very powerful adversary who is able to capture tens of thousands of nodes.

References

Advanced Simulation and Computing. (2012, October). *Sustainable stockpile stewardship.* Retrieved from `https://asc.llnl.gov/computing\_resources/sequoia/`

Barker, E., Barker, W. B., Burr, W., Polk, W., & Smid, M. (2012, July). *Recommendation for Key Management – Part 1: General* (No. Special Publication 800-57). Gaithersburg, MD 20899-8930.

Blom, R. (1983). Non-Public Key Distribution. *Advances in Cryptology, Proceedings of Crypto '82*, 231–236.

Blom, R. (1984). *An Optimal Class of Symmetric Key Generation Systems* (Tech. Rep.). Linkopping University, Linkopping, Sweden.

Blundo, C., De Santis, A., Herzberg, A., Kutten, S., Vaccaro, U., & Yung, M. (1995, June). *Perfectly-Secure Key Distribution for Dynamic Conferences* (Tech. Rep.). Dipartimento di Informatica ed Applicazioni, Universita di Salerno, 84081 Baronissi, Italy.

Boneh, D. (1998). The decision diffie-hellman problem. In *The third algorithmic number theory symposium* (Vol. 1423, pp. 48–63). Springer-Verlag.

Boneh, D., & Franklin, M. (2001). Identity-Based Encryption from the Weil Pairing. *Proceedings of CRYPTO 2001, LNCS 2139*, 213–229.

Camtepe, Seyit Ahmet and Yener, Bülent. (2007). Combinatorial design of key distribution mechanisms for wireless sensor networks. *IEEE/ACM Transactions on Networking*, *15*(2), 346–358.

CC2420 Data Sheet [Computer software manual]. (2014). Retrieved from `http://focus.ti.com/lit/ds/symlink/cc2420.pdf`

Chan, H., & Perrig, A. (2005). Pike: Peer intermediaries for key establishment in sensor networks. *In Proceedings of IEEE Infocom*, 524–535.

Chan, H., Perrig, A., & Song, D. (2003). Random key predistribution schemes for sensor networks. In *Proceedings of the 2003 ieee symposium on security and privacy* (pp. 197–). Washington, DC, USA: IEEE Computer Society. Retrieved from `http://portal.acm.org/citation.cfm?id=829515.830566`

Chen, N., Yao, J.-b., & Wen, G.-j. (2008, 29-31 July). An Improved Matrix Key Pre-distribution Scheme for Wireless Sensor Networks. *International Conference on Embedded Software Systems*, 40–45.

Cheng, H.-b., Yang, G., Wang, J.-t., & Huang, X. (2006, Jun). An authenticated identity-based key establishment and encryption scheme for wireless sensor networks.

The Journal of China Universities of Posts adn Telecommunications, *13*(2).

Chien, H. Y., Chen, R.-C., & Shen, A. (2008). Efficient key pre-distribution for sensor nodes with strong connectivity and low storage space. *Poceedings of the 22nd International Conference on Advanced Information Networking and Applications (AINA'08), IEEE Computer Society, Washington, DC, USA.*, 327-333.

Crosby, S., Goldberg, I., Johnson, R., Song, D., & Wagner, D. (2001, May). A Cryptanalysis of the High-bandwidth Digital Content Protection System. *ACM-CCS8 DRM Workshop*.

de Meulenaer, G., Gosset, F., Standaert, F.-X., & Pereira, O. (2008, 10). On the energy cost of communications and cryptography in wireless sensor networks. *IEEE International Conference on Wireless & Mobile Computing, Networking & Communication*, 580–585.

Diffie, W., & Hellman, M. (1976, Nov). New directions in cryptography. *IEEE Transactions on Information Theory*, *22*(6), 644-654. doi: 10.1109/TIT.1976.1055638

Dolev, D., & Yao, A. C. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, *IT-29*, 198–208.

Du, W., Han, S. Y., Deng, J., & Varshney, P. K. (2003, October). A pairwise key pre-distribution scheme for wireless sensor networks. *Proceedings of the conference on Computer and communications security*.

Du, W., Wang, R., & Ning, P. (2005). An efficient scheme for authenticating public keys in sensor networks. *6th ACM international symposium on Mobile ad hoc networking and computing*, 58-67.

Elgamal, T. (1985, Jul). A public key cryptosystem and a signature scheme based on discrete logarithms. *Information Theory, IEEE Transactions on*, *31*(4), 469–472. doi: 10.1109/TIT.1985.1057074

Eschenauer, L., & Gligor, V. D. (2002). A key-management scheme for distributed sensor networks. In (pp. 41–47). New York, NY, USA: ACM Press.

Fan, Y., Chen, I.-R., & Eltoweissy, M. (2005). On Optimal Key Disclosure Interval for μTESLA: Analysis of Authentication Delay versus Network Cost. *International Conference on Wireless Networks, Communication and Mobile Computing*, 304 - 309.

Fiore, D., & Gennaro, R. (2010). Making the diffie-hellman protocol identity-based. *CT-RSA 2010, LNCS*.

Gay, D., Welsh, M., Levis, P., Brewer, E., Behren, R. V., & Culler, D. (2003). The nesc language: A holistic approach to networked embedded systems. In *In Proceedings of Programming Language Design and Implementation (PLDI)* (pp. 1–11).

Grosschadl, J., Szekely, A., & Tillich, S. (2007). The energy cost of cryptographic key establishment in wireless sensor networks. *Proc. The 2nd ACM Symposium on Information, Compter and Communication Security*.

Gura, N., Patel, A., Wander, A., Eberle, H., & Shantz, S. C. (2004, August). Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. *In Proceedings of the 2004 Workshop on Cryptographic Hardware and Embedded Systems*.

Hartung, C., Balasalle, J., & Han, R. (2005, January). *Node compromise in sensor*

networks: The need for secure systems (Tech. Rep.). Department of Computer Science, University of Colorado at Boulder.

Hayashi, T., Shimoyama, T., Shinohara, N., & Takagi, T. (2012). *Breaking pairing-based cryptosystems using η_t pairing over $gf(3^{97})$*. Cryptology ePrint Archive, Report 2012/345. (`http://eprint.iacr.org/`)

IEEE. (2006, September). *IEEE Standard 802.15.4 (2006) Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)* (No. 802.15.4). 3 Park Avenue, New York.

IEEE Computer Society. (2006, September). Ieee std 802.15.4-2006.

Karlof, C., Sastry, N., & Wagner, D. (2004, November). Tinysec: A link layer security architecture for wireless sensor networks. *Proc. of the Second ACM Conference on Embedded Networked Sensor Systems (Sensys 2004)*. (TinySec)

Law, L. E., Menezes, A. J., Qu, M., Solinas, J. A., & Vanstone, S. A. (2003). An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, *28*(2), 119–134.

Lederer, C., Mader, R., Koschuch, M., Grosschadl, J., Szekely, A., & Tillich, S. (2009). *Energy-Efficient Implementation of ECDH Key Exchange for Wireless Sensor Networks*. Springer Verlag, LNCS 5746.

Lee, H.-S., Lee, Y.-R., & Lee, J.-H. (2003). Multiparty Key Agreement Protocol based on Symmetric Techniques. *Commun. Korean Math. Soc.*, *18*(1), 169–179.

Lee, J., & Stinson, D. R. (2005). Deterministic Key Predistribution Schemes for Distributed Sensor Networks. In H. Handschuh & A. Hasan (Eds.), (Vol. 3357, pp. 294–307). Springer-Verlag Berlin Heidelberg.

Leighton, F. T., & Micali, S. (1994). Secret-Key Agreement without Public-Key Cryptography. In *Proceedings of the 13th annual international cryptology conference on advances in cryptology* (pp. 456–479). London, UK, UK: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=646758.705685`

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., ... others (2005). Tinyos: An operating system for sensor networks.

Liu, A., & Ning, P. (2008, April). TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. *in Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, 245–256.

Liu, D., Ning, P., & Li, R. (2003). Establishing Pairwise Keys in Distributed Sensor Networks. *Proceedings of the 10th ACM conference on Computer and communications security*.

Liu, M., Wei, W., & Liu, Z. (2009, May). A secure key pre-distribution scheme for wireless sensor networks. *International Conference on Industrial Electronics and Applications, ICIEA.*, 1762 –1768. doi: 10.1109/ICIEA.2009.5138499

Luk, M., Mezzour, G., Perrig, A., & Gilgor, V. (2007, April). Minisec: A secure sensor network communication architecture. *IPSN*.

Memsic Corp. (n.d.). Memsic corporation, micaz datasheet [Computer software manual]. Retrieved from `http://www.memsic.com/products/wireless`

-sensor-networks/wireless-modules.html,Retrieved: April12,2012

Memsic Corp. (2012). MICAz Datasheet [Computer software manual]. Retrieved from http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html,Retrieved: April12,2012

Menezes, A., van Oorschot, P., & Vanstone, S. (1996). *Handbook of applied cryptography*. CRC Press, Inc.

Menezes, A. J., Oorschot, P. C., & Vanston, S. A. (2001). *Handbook of applied cryptography*. CRC Press, Inc.

National Security Agency. (2009, Jan). *The Case for Elliptic Curve Cryptography.* Retrieved from http://www.nsa.gov/business/programs/elliptic_curve.shtml

Oliveira, L., Scott, M., Lopez, J., & Dahab, R. (2008, June). TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Networked Sensing Systems, 2008. INSS 2008. 5th International Conference on*, 173 –180. doi: 10.1109/INSS.2008.4610921

Oliveira, L. B., Aranha, D. F., Gouvêa, C. P. L., Scott, M., Címara, D. F., López, J., & Dahab, R. (2011, March). TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Computer Communications*, *34*, 485–493. Retrieved from http://dx.doi.org/10.1016/j.comcom.2010.05.013 doi: http://dx.doi.org/10.1016/j.comcom.2010.05.013

Oliveira, L. B., Aranha, D. F., Morais, E., Daguano, F., López, J., & Dahab, R. (2007). Tinytate: Identity-based encryption for sensor networks. *IACR Cryptology ePrint Archive*, *2007*.

P. Levis. (2006). *TinyOS programming.* Retrieved from http://csl.stanford.edu/$\sim$pal/pubs/tinyos-programming.pdf.Retrieved: April12,2012

Paar, C., & Pelzl, J. (2010). *Understanding Cryptography*. Springer Heidelberg Dordrecht.

Perrig, A., Canetti, R., Tygar, J. D., & Song, D. (2002, Summer/Fall). The TESLA Broadcast Authentication Protocol. *CryptoBytes*, *5*(2), 2–13.

Perrig, A., Szewczyk, R., Wen, V., Culler, D., & Tygar, J. D. (2002). SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, *8*, 521–534.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120–126.

Schnorr, C. (n.d.). Efficient identification and signatures for smart cards. In G. Brassard (Ed.), *Lncs* (Vol. 435, p. 239-252). Springer, Heidelberg (1990).

Shamir, A. (1985). Identity-based Cryptography and Signature Schemes. *Proceedings of CRYPTO'84, LNCS 196*, 47–53.

Steiner, J. G., Neuman, C., & Schiller, J. I. (1988, February). Kerberos: An authentication service for open network systems. *In Proceedings of the Winter 1988 USENIX Conference. USENIX*.

Stinson, D. R. (2006). *Cryptography theory and practice* (Third ed.; H. R. Kenneth, Ed.). Chapman & Hall/CRC.

Szczechowiak, P., & Collier, M. (2009, Aug.). Practical identity-based key agreement for secure communication in sensor networks. , 1–6. doi: 10.1109/ICCCN.2009.5235277

Tapia, A. R., Lanius, C., Mac Zeal, C. M., & Parks, T. A. (2001). Computational science: Tools for a changing world. *http://ceee.rice.edu/Books/CS/chapter5/cost1.html*.

Ugus, O., Westhoff, D., Laue, R., Shoufan, A., & Huss, S. A. (2007, October). Optimized Implementation of Elliptic Curve Based Additive Homomorphic Encryption for Wireless Sensor Networks. *2nd Workshop on Embedded Systems Security - WESS'2007, Salzburg, Austria.*.

Wang, H., & Li, Q. (2006). Efficient implementation of public key cryptosystems on mote sensors (short paper). In *In international conference on information and communication security (icics), lncs 4307* (pp. 519–528).

Wang, S.-J., Tsai, Y.-R., & Chan, J.-W. (2007). A countermeasure against frequent attacks based on the blom-scheme in ad hoc sensor networks. *International Symposium on Wireless Pervasive Computing*.

Watro, R., Kong, D., Cuti, S.-f., Gardiner, C., Lyn, C., & Kruus, P. (2004). TinyPK: Securing Sensor Networks with Public Key Technology. *Proc. 2nd ACM Workshop on Security of ad hoc and sensor networks*.

Yu, C.-M., Lu, C.-S., & Kuo, S.-Y. (2010). Noninteractive Pairwise Key Establishment for Sensor Networks. *IEEE Transactions on Information Forensics and Security*, *5*(3), 556–569.

Zhang, J., Yu, W., & Liu, X. (2009, January). CRTBA: Chinese Remainder Theorem-Based Broadcast Authentication in Wireless Sensor Network. *International Symposium on Computer Network and Multmedia Technology*, 1–4. Retrieved from `http://dx.doi.org/10.1109/CNMT.2009.5374495` (CRT) doi: 10.1109/CNMT.2009.5374495

Zhang, W., Zhu, S., & Cao, G. (2007, September). A Random Perturbation-Based Scheme for Pairwise Key Establishment in Sensor Networks. *MobiHoc'07*.

Zheng, H., & Dai, H. (2008, sept.). A polynomial-based key predistribution scheme for wireless sensor networks using matrix decomposition. , 1022 –1026. doi: 10.1109/ICCIS.2008.4670785

Zhou, J., & He, M. (2009). An Improved Distributed Key Mangement Scheme in Wireless Sensor Networks. *WISA 2008, LNCS 5379*, 305–319.

Zhu, S., Setia, S., & Jajodia, S. (2003). LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks. *Proceedings of the 10th ACM conference on Computer and communications security*.

Zhu, S., Setia, S., & Jajodia, S. (2006, November). LEAP+: Efficient Security Mechanisms for Large-Scale Distributed Sensor Network. *ACM Transactions on Sensor Networks*, *2*(4), 500–528.

ZigBee, A. (2005). *Zigbee specifications* [ZigBee Specifications].

Appendix A

Design and Performance Tables

These values are obtained for the MICAz mote running the TinyOS code in Appendix B.4. No attempt was made to optimise the code for speed and efficiency.

η	N	Q_R	$m = 12$		$m = 16$		$m = 24$		$m = 32$	
			Q_o	T_{comp}	Q_o	T_{comp}	Q_o	T_{comp}	Q_o	T_{comp}
3	3	27	108	38.87	144	44.01	216	54.28	288	64.55
	4	36	144	43.49	192	50.17	288	63.52	384	76.88
	5	45	180	48.12	240	56.33	360	72.77	480	89.20
	6	54	216	52.74	288	62.50	432	82.01	576	101.53
	7	63	252	57.36	336	68.66	504	91.26	672	113.86
	8	72	288	61.98	384	74.82	576	100.50	768	126.18
4	3	48	144	50.09	192	58.99	288	76.79	384	94.60
	4	64	192	58.30	256	69.94	384	93.23	512	116.51
	5	80	240	66.52	320	80.90	480	109.66	640	138.42
	6	96	288	74.74	384	91.86	576	126.10	768	160.34
	7	112	336	82.96	448	102.81	672	142.53	896	182.25
	8	128	384	91.17	512	113.77	768	158.97	1024	204.16
5	3	75	180	64.38	240	78.08	360	105.47	480	132.86
	4	100	240	77.22	320	95.20	480	131.15	640	167.10
	5	125	300	90.06	400	112.32	600	156.83	800	201.34
	6	150	360	102.90	480	129.44	720	182.51	960	235.58
	7	175	420	115.74	560	146.56	840	208.19	1120	269.82
	8	200	480	128.58	640	163.68	960	233.87	1280	304.06
6	4	144	288	100.25	384	125.93	576	177.29	768	228.65
	5	180	360	118.74	480	150.58	720	214.27	960	277.95
	6	216	432	137.23	576	175.23	864	251.24	1152	327.26
	7	252	504	155.72	672	199.88	1008	288.22	1344	376.56
	8	288	576	174.20	768	224.54	1152	325.20	1536	425.87
7	3	147	252	102.22	336	128.58	504	181.31	672	234.04
	4	196	336	127.38	448	162.14	672	231.64	896	301.15
	5	245	420	152.55	560	195.69	840	281.98	1120	368.26
	6	294	504	177.71	672	229.25	1008	332.31	1344	435.37
	7	343	588	202.88	784	262.80	1176	382.64	1568	502.48
	8	392	672	228.05	896	296.36	1344	432.97	1792	569.59
8	3	192	288	125.76	384	160.00	576	228.48	768	296.96
	4	256	384	158.63	512	203.82	768	294.22	1024	384.61
	5	320	480	191.50	640	247.65	960	359.96	1280	472.26
	6	384	576	224.37	768	291.48	1152	425.70	1536	559.92
	7	448	672	257.24	896	335.30	1344	491.44	1792	647.57
	8	512	768	290.11	1024	379.13	1536	557.18	2048	735.23

Key sizes 64 bits 80 bits 96 bits 112 bits 128 bits

Table A.1: Performance – RAM, ROM, and Computation Times

Prime modulus $p = 31$												
η	N	Key size (bits)	Capture n_c (nodes)	$m = 12$			$m = 16$			$m = 24$		
				Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)	Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)	Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)
3	3	54	2	7.22	108	38.87	9.63	144	44.01	14.45	216	54.28
	4	62	2	9.34	144	43.49	12.45	192	50.17	18.68	288	63.52
	5	69	4	11.45	180	48.12	15.27	240	56.33	22.90	360	72.77
	6	76	10	12.95	216	52.74	17.27	288	62.50	25.90	432	82.01
	7	81	28	13.75	252	57.36	18.34	336	68.66	27.51	504	91.26
	8	86	92	14.77	288	61.98	19.69	384	74.82	29.53	576	100.50
4	3	72	2	9.34	144	50.09	12.45	192	58.99	18.68	288	76.79
	4	81	6	11.45	192	58.30	15.27	256	69.94	22.90	384	93.23
	5	89	23	13.37	240	66.52	17.82	320	80.90	26.74	480	109.66
	6	96	128	14.45	288	74.74	19.27	384	91.86	28.90	576	126.10
	7	102	845	15.61	336	82.96	20.82	448	102.81	31.23	672	142.53
	8	107	6318	16.56	384	91.17	22.08	512	113.77	33.13	768	158.97
5	3	87	4	10.84	180	64.38	14.45	240	78.08	21.67	360	105.47
	4	98	24	12.95	240	77.22	17.27	320	95.20	25.90	480	131.15
	5	106	238	14.45	300	90.06	19.27	400	112.32	28.90	600	156.83
	6	113	3011	15.87	360	102.90	21.16	480	129.44	31.73	720	182.51
	7	119	4.51×10^4	16.78	420	115.74	22.37	560	146.56	33.55	840	208.19
	8	125	7.58×10^5	17.73	480	128.58	23.63	640	163.68	35.45	960	233.87
6	3	101	11	12.00	216	81.76	16.00	288	101.27	24.00	432	140.31
	4	112	156	14.11	288	100.25	18.82	384	125.93	28.23	576	177.29
	5	121	3486	15.61	360	118.74	20.82	480	150.58	31.23	720	214.27
	6	128	9.96×10^4	16.98	432	137.23	22.64	576	175.23	33.96	864	251.24
	7	134	3.32×10^6	17.90	504	155.72	23.86	672	199.88	35.79	1008	288.22
	8	140	1.22×10^8	18.68	576	174.20	24.90	768	224.54	37.35	1152	325.20
	9	145	4.81×10^9	19.35	648	192.70	25.80	864	249.20	38.71	1296	362.20
7	3	113	38	12.95	252	102.22	17.27	336	128.58	25.90	504	181.31
	4	124	1267	15.06	336	127.38	20.08	448	162.14	30.13	672	231.64
	5	133	6.41×10^4	16.56	420	152.55	22.08	560	195.69	33.13	840	281.98
	6	140	4.07×10^6	17.73	504	177.71	23.63	672	229.25	35.45	1008	332.31
	7	147	2.95×10^8	18.68	588	202.88	24.90	784	262.80	37.35	1176	382.64
	8	152	2.32×10^{10}	19.48	672	228.05	25.97	896	296.36	38.96	1344	432.97
8	3	123	158	13.75	288	125.76	18.34	384	160.00	27.51	576	228.48
	4	135	1.22×10^4	15.87	384	158.63	21.16	512	203.82	31.73	768	294.22
	5	144	1.37×10^6	17.37	480	191.50	23.16	640	247.65	34.73	960	359.96
	6	152	1.90×10^8	18.53	576	224.37	24.71	768	291.48	37.06	1152	425.70
	7	158	2.94×10^{10}	19.48	672	257.24	25.97	896	335.30	38.96	1344	491.44
	8	164	4.86×10^{12}	20.28	768	290.11	27.04	1024	379.13	40.57	1536	557.18

Key sizes 64 bits 80 bits 96 bits 112 bits 128 bits

Table A.2: Security, Resilience, and Performance Table. Traitor node capture n_c, Number of Master Key Solutions Φ in 10^x

Prime modulus $p = 23$												
η	N	Key size (bits)	Capture n_c (nodes)	$m = 12$			$m = 16$			$m = 24$		
				Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)	Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)	Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)
3	3	45	2	7.22	108	38.87	9.63	144	44.01	14.45	216	54.28
	4	52	3	10.14	144	43.49	13.52	192	50.17	20.28	288	63.52
	5	58	8	12.00	180	48.12	16.00	240	56.33	24.00	360	72.77
	6	63	25	13.37	216	52.74	17.82	288	62.50	26.74	432	82.01
	7	67	90	14.45	252	57.36	19.27	336	68.66	28.90	504	91.26
	8	70	365	15.06	288	61.98	20.08	384	74.82	30.13	576	100.50
4	3	60	3	9.34	144	50.09	12.45	192	58.99	18.68	288	76.79
	4	67	12	12.00	192	58.30	16.00	256	69.94	24.00	384	93.23
	5	73	75	13.75	240	66.52	18.34	320	80.90	27.51	480	109.66
	6	78	557	14.77	288	74.74	19.69	384	91.86	29.53	576	126.10
	7	83	4697	15.87	336	82.96	21.16	448	102.81	31.73	672	142.53
	8	87	4.31×10^4	16.78	384	91.17	22.37	512	113.77	33.55	768	158.97
5	3	72	8	11.45	180	64.38	15.27	240	78.08	22.90	360	105.47
	4	80	83	13.37	240	77.22	17.82	320	95.20	26.74	480	131.15
	5	86	1164	15.06	300	90.06	20.08	400	112.32	30.13	600	156.83
	6	91	1.96×10^4	16.11	360	102.90	21.48	480	129.44	32.22	720	182.51
	7	96	3.67×10^5	17.18	420	115.74	22.90	560	146.56	34.35	840	208.19
	8	100	7.31×10^6	17.90	480	128.58	23.86	640	163.68	35.79	960	233.87
6	3	82	31	12.50	216	81.76	16.66	288	101.27	24.99	432	140.31
	4	90	737	14.45	288	100.25	19.27	384	125.93	28.90	576	177.29
	5	97	2.34×10^4	16.11	360	118.74	21.48	480	150.58	32.22	720	214.27
	6	102	8.66×10^5	17.18	432	137.23	22.90	576	175.23	34.35	864	251.24
	7	107	3.48×10^7	18.06	504	155.72	24.08	672	199.88	36.12	1008	288.22
	8	111	1.47×10^9	18.96	576	174.20	25.28	768	224.54	37.92	1152	325.20
7	3	91	145	13.37	252	102.22	17.82	336	128.58	26.74	504	181.31
	4	99	7836	15.35	336	127.38	20.46	448	162.14	30.69	672	231.64
	5	106	5.48×10^5	16.98	420	152.55	22.64	560	195.69	33.96	840	281.98
	6	112	4.34×10^7	18.06	504	177.71	24.08	672	229.25	36.12	1008	332.31
	7	116	3.66×10^9	18.96	588	202.88	25.28	784	262.80	37.92	1176	382.64
	8	121	3.19×10^{11}	19.72	672	228.05	26.30	896	296.36	39.44	1344	432.97
8	3	99	773	14.45	288	125.76	19.27	384	160.00	28.90	576	228.48
	4	107	9.32×10^4	16.34	384	158.63	21.79	512	203.82	32.68	768	294.22
	5	114	1.40×10^7	17.55	480	191.50	23.40	640	247.65	35.10	960	359.96
	6	120	2.33×10^9	18.68	576	224.37	24.90	768	291.48	37.35	1152	425.70
	7	125	4.07×10^{11}	19.60	672	257.24	26.14	896	335.30	39.20	1344	491.44
	8	129	7.26×10^{13}	20.39	768	290.11	27.18	1024	379.13	40.78	1536	557.18

Key sizes 64 bits 80 bits 96 bits 112 bits 128 bits

Table A.3: Security and Performance Features using $m = 16$. Traitor Node Capture n_c and Number of Master Key Solutions Φ are 10^x

Prime modulus $p = 19$												
η	N	Key size (bits)	Capture n_c (nodes)	$m = 12$			$m = 16$			$m = 24$		
				Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)	Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)	Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)
3	3	41	2	8.39	108	38.87	11.18	144	44.01	16.78	216	54.28
	4	46	4	10.14	144	43.49	13.52	192	50.17	20.28	288	63.52
	5	51	13	12.00	180	48.12	16.00	240	56.33	24.00	360	72.77
	6	55	48	13.37	216	52.74	17.82	288	62.50	26.74	432	82.01
	7	59	198	14.45	252	57.36	19.27	336	68.66	28.90	504	91.26
	8	62	891	15.35	288	61.98	20.46	384	74.82	30.69	576	100.50
4	3	53	4	10.14	144	50.09	13.52	192	58.99	20.28	288	76.79
	4	59	23	12.50	192	58.30	16.66	256	69.94	24.99	384	93.23
	5	64	166	13.75	240	66.52	18.34	320	80.90	27.51	480	109.66
	6	68	1428	15.06	288	74.74	20.08	384	91.86	30.13	576	126.10
	7	72	1.35×10^4	16.11	336	82.96	21.48	448	102.81	32.22	672	142.53
	8	75	1.35×10^5	16.98	384	91.17	22.64	512	113.77	33.96	768	158.97
5	3	63	15	11.45	180	64.38	15.27	240	78.08	22.90	360	105.47
	4	69	188	13.75	240	77.22	18.34	320	95.20	27.51	480	131.15
	5	75	3159	15.06	300	90.06	20.08	400	112.32	30.13	600	156.83
	6	79	6.02×10^4	16.34	360	102.90	21.79	480	129.44	32.68	720	182.51
	7	83	1.23×10^6	17.18	420	115.74	22.90	560	146.56	34.35	840	208.19
	8	86	2.61×10^7	18.06	480	128.58	24.08	640	163.68	36.12	960	233.87
6	3	71	66	12.95	216	81.76	17.27	288	101.27	25.90	432	140.31
	4	78	1970	14.77	288	100.25	19.69	384	125.93	29.53	576	177.29
	5	84	7.30×10^4	16.11	360	118.74	21.48	480	150.58	32.22	720	214.27
	6	88	2.98×10^6	17.37	432	137.23	23.16	576	175.23	34.73	864	251.24
	7	92	1.28×10^8	18.22	504	155.72	24.30	672	199.88	36.44	1008	288.22
	8	95	5.63×10^9	18.96	576	174.20	25.28	768	224.54	37.92	1152	325.20
7	3	79	352	13.75	252	102.22	18.34	336	128.58	27.51	504	181.31
	4	86	2.36×10^4	15.61	336	127.38	20.82	448	162.14	31.23	672	231.64
	5	91	1.87×10^6	16.98	420	152.55	22.64	560	195.69	33.96	840	281.98
	6	96	1.60×10^8	18.06	504	177.71	24.08	672	229.25	36.12	1008	332.31
	7	100	1.42×10^{10}	18.96	588	202.88	25.28	784	262.80	37.92	1176	382.64
	8	103	1.27×10^{12}	19.84	672	228.05	26.45	896	296.36	39.68	1344	432.97
8	3	85	2104	14.45	288	125.76	19.27	384	160.00	28.90	576	228.48
	4	92	3.06×10^5	16.34	384	158.63	21.79	512	203.82	32.68	768	294.22
	5	98	5.12×10^7	17.73	480	191.50	23.63	640	247.65	35.45	960	359.96
	6	103	9.04×10^9	18.82	576	224.37	25.09	768	291.48	37.64	1152	425.70
	7	107	1.63×10^{12}	19.72	672	257.24	26.30	896	335.30	39.44	1344	491.44
	8	110	2.96×10^{14}	20.49	768	290.11	27.32	1024	379.13	40.98	1536	557.18

Key Sizes 64 bits 80 bits 96 bits 112 bits 128 bits

Table A.4: Security and Performance Features Using $m = 16$. Traitor Node Capture n_c and Number of Master Key Solutions Φ are 10^x

Prime modulus $p = 17$												
η	N	Key size (bits)	Capture n_c (nodes)	$m = 12$			$m = 16$			$m = 24$		
				Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)	Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)	Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)
3	3	38	2	8.39	108	38.87	11.18	144	44.01	16.78	216	54.28
	4	43	5	10.84	144	43.49	14.45	192	50.17	21.67	288	63.52
	5	48	18	12.50	180	48.12	16.66	240	56.33	24.99	360	72.77
	6	51	71	13.75	216	52.74	18.34	288	62.50	27.51	432	82.01
	7	54	313	14.45	252	57.36	19.27	336	68.66	28.90	504	91.26
	8	57	1482	15.35	288	61.98	20.46	384	74.82	30.69	576	100.50
4	3	49	5	10.14	144	50.09	13.52	192	58.99	20.28	288	76.79
	4	55	33	12.50	192	58.30	16.66	256	69.94	24.99	384	93.23
	5	59	264	14.11	240	66.52	18.82	320	80.90	28.23	480	109.66
	6	63	2429	15.06	288	74.74	20.08	384	91.86	30.13	576	126.10
	7	66	2.41×10^4	16.11	336	82.96	21.48	448	102.81	32.22	672	142.53
	8	69	2.49×10^5	16.98	384	91.17	22.64	512	113.77	33.96	768	158.97
5	3	58	21	12.00	180	64.38	16.00	240	78.08	24.00	360	105.47
	4	64	303	13.75	240	77.22	18.34	320	95.20	27.51	480	131.15
	5	69	5511	15.35	300	90.06	20.46	400	112.32	30.69	600	156.83
	6	73	1.11×10^5	16.34	360	102.90	21.79	480	129.44	32.68	720	182.51
	7	76	2.35×10^6	17.37	420	115.74	23.16	560	146.56	34.73	840	208.19
	8	79	5.10×10^7	18.06	480	128.58	24.08	640	163.68	36.12	960	233.87
6	3	66	102	12.95	216	81.76	17.27	288	101.27	25.90	432	140.31
	4	72	3416	14.77	288	100.25	19.69	384	125.93	29.53	576	177.29
	5	77	1.35×10^5	16.34	360	118.74	21.79	480	150.58	32.68	720	214.27
	6	81	5.74×10^6	17.37	432	137.23	23.16	576	175.23	34.73	864	251.24
	7	84	2.53×10^8	18.22	504	155.72	24.30	672	199.88	36.44	1008	288.22
	8	87	1.13×10^{10}	19.09	576	174.20	25.46	768	224.54	38.19	1152	325.20
7	3	72	585	13.75	252	102.22	18.34	336	128.58	27.51	504	181.31
	4	79	4.31×10^4	15.87	336	127.38	21.16	448	162.14	31.73	672	231.64
	5	84	3.60×10^6	17.18	420	152.55	22.90	560	195.69	34.35	840	281.98
	6	88	3.17×10^8	18.22	504	177.71	24.30	672	229.25	36.44	1008	332.31
	7	91	2.85×10^{10}	19.09	588	202.88	25.46	784	262.80	38.19	1176	382.64
	8	94	2.58×10^{12}	19.84	672	228.05	26.45	896	296.36	39.68	1344	432.97
8	3	78	3677	14.77	288	125.76	19.69	384	160.00	29.53	576	228.48
	4	85	5.79×10^5	16.56	384	158.63	22.08	512	203.82	33.13	768	294.22
	5	90	1.01×10^8	17.90	480	191.50	23.86	640	247.65	35.79	960	359.96
	6	94	1.82×10^{10}	18.82	576	224.37	25.09	768	291.48	37.64	1152	425.70
	7	97	3.30×10^{12}	19.72	672	257.24	26.30	896	335.30	39.44	1344	491.44
	8	100	6.03×10^{14}	20.49	768	290.11	27.32	1024	379.13	40.98	1536	557.18

Key sizes: 64 bits, 80 bits, 96 bits, 112 bits, 128 bits

Table A.5: Security and Performance Features using $m = 16$. Traitor Node Capture n_c and Number of Master Key Solutions Φ are 10^x

Prime modulus $p = 13$												
η	N	Key size (bits)	Capture n_c (nodes)	$m = 12$			$m = 16$			$m = 24$		
				Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)	Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)	Φ (10^x)	Q_o (Bytes)	T_{comp} (ms)
3	3	32	3	8.39	108	38.87	11.18	144	44.01	16.78	216	54.28
	4	36	10	10.84	144	43.49	14.45	192	50.17	21.67	288	63.52
	5	39	39	12.50	180	48.12	16.66	240	56.33	24.99	360	72.77
	6	42	182	13.75	216	52.74	18.34	288	62.50	27.51	432	82.01
	7	45	907	14.77	252	57.36	19.69	336	68.66	29.53	504	91.26
	8	47	4677	15.61	288	61.98	20.82	384	74.82	31.23	576	100.50
4	3	40	10	10.84	144	50.09	14.45	192	58.99	21.67	288	76.79
	4	45	80	12.95	192	58.30	17.27	256	69.94	25.90	384	93.23
	5	48	772	14.11	240	66.52	18.82	320	80.90	28.23	480	109.66
	6	51	7978	15.35	288	74.74	20.46	384	91.86	30.69	576	126.10
	7	54	8.55×10^4	16.34	336	82.96	21.79	448	102.81	32.68	672	142.53
	8	56	9.34×10^5	16.98	384	91.17	22.64	512	113.77	33.96	768	158.97
5	3	47	49	12.00	180	64.38	16.00	240	78.08	24.00	360	105.47
	4	52	907	14.11	240	77.22	18.82	320	95.20	28.23	480	131.15
	5	56	1.89×10^4	15.35	300	90.06	20.46	400	112.32	30.69	600	156.83
	6	59	4.12×10^5	16.56	360	102.90	22.08	480	129.44	33.13	720	182.51
	7	61	9.16×10^6	17.37	420	115.74	23.16	560	146.56	34.73	840	208.19
	8	63	2.05×10^8	18.22	480	128.58	24.30	640	163.68	36.44	960	233.87
6	3	53	287	13.37	216	81.76	17.82	288	101.27	26.74	432	140.31
	4	58	1.16×10^4	15.06	288	100.25	20.08	384	125.93	30.13	576	177.29
	5	62	5.07×10^5	16.34	360	118.74	21.79	480	150.58	32.68	720	214.27
	6	65	2.28×10^7	17.55	432	137.23	23.40	576	175.23	35.10	864	251.24
	7	67	1.03×10^9	18.38	504	155.72	24.50	672	199.88	36.76	1008	288.22
	8	70	4.68×10^{10}	19.09	576	174.20	25.46	768	224.54	38.19	1152	325.20
7	3	58	1854	14.11	252	102.22	18.82	336	128.58	28.23	504	181.31
	4	63	1.58×10^5	15.87	336	127.38	21.16	448	162.14	31.73	672	231.64
	5	67	1.42×10^7	17.18	420	152.55	22.90	560	195.69	34.35	840	281.98
	6	70	1.30×10^9	18.22	504	177.71	24.30	672	229.25	36.44	1008	332.31
	7	73	1.19×10^{11}	19.09	588	202.88	25.46	784	262.80	38.19	1176	382.64
	8	75	1.09×10^{13}	19.95	672	228.05	26.60	896	296.36	39.91	1344	432.97
8	3	63	1.27×10^4	14.77	288	125.76	19.69	384	160.00	29.53	576	228.48
	4	68	2.24×10^6	16.56	384	158.63	22.08	512	203.82	33.13	768	294.22
	5	71	4.11×10^8	17.90	480	191.50	23.86	640	247.65	35.79	960	359.96
	6	74	7.57×10^{10}	18.96	576	224.37	25.28	768	291.48	37.92	1152	425.70
	7	77	1.39×10^{13}	19.84	672	257.24	26.45	896	335.30	39.68	1344	491.44
	8	79	2.55×10^{15}	20.59	768	290.11	27.46	1024	379.13	41.18	1536	557.18

Key sizes 64 bits 80 bits 96 bits 112 bits 128 bits

Table A.6: Security and Performance Features using $m = 16$. Traitor Node Capture n_c and Number of Master Key Solutions Φ are 10^x

Appendix B

SOURCE CODES

B.1 Simulate Probability of Finding a Traitor Node

```
% ==========================================
% PROBABILITY OF FINDING THE PILOT NODE
%==========================================
% FILL THE n COUPLERS WITH RANDOM NUM
clear;
pr = 7;
N = 4   ;
eta =3;
Na=N*eta-N;
txteta = int2str(eta);
txtN = int2str(N);
txtpr = int2str(pr);
% fName = strcat("./results/findRefNode_probab","_N",txtN,"_e",txteta,"_pr",txtpr);
runs = 1e10;
ctr=0;
foundRef = false;
for r=1:runs
  for i=1:N
    couplers(i) = 1+floor(rand()*pr);
  end
% FILL SUBSET Ra WITH RANDOM NUMBERS
  for i=1:(Na)
    Ra(i) = 1+floor(rand()*pr);
  end
% FILL SUBSET Rb WITH RANDOM NUMBERS
  for i=1:(Na)
    Rb(i)= 1+floor(rand()*pr);
  end
% CHECK IF Ra INTERSECTS (Rb UNION COUPLERS)
  RbC = union(Rb,couplers);
  RaC = union(Ra,couplers);
  tfa = intersect(Ra,RbC);
  tfb = intersect(Rb,RaC);
  if ((size(tfa,2) == 0)  || (size(tfb,2) == 0))
    % foundRef = true;
    disp('========= FOUND ======')
    size(tfa,2);
    size(tfb,2);
```

```
    ctr=ctr+1;
    %  disp("*********");disp(couplers);
    % disp("---------");disp(" tfa: ");disp(tfa);disp("Ra:");disp(Ra);disp("RbC");disp(RbC);
    % disp("+++++++++");disp(" tfb: ");disp(tfb);disp("Rb:");disp(Rb);disp("RaC");disp(RaC);
  else
    foundRef= false;
  end
  % save(fName,'r','ctr','N','eta','pr')
end
disp('prime ');disp(pr);
disp('N ');disp(N);
disp('eta' );disp(eta);
disp('found = ');disp(ctr);
disp('Probab = ');disp(ctr/runs)
% disp("Expected: ");disp(runs/ctr);
```

B.2 Simulate Capture to Find a Traitor Node

```
%=========================================================================
% SIMULATE NODE CAPTURE TO FIND the TRAITOR NODE
% COUNT SUCCESS AND RUNS, PROBAB = SUCCESS/RUNS
% ALSO ACCUMULATE THE NUMBER OF COUPLINGS Ra IN EACH RUN TO GET DIST.
% COMPUTE THE KEY-SET USING ACTUAL MASTER KEYS, PUBLIC KEYS
% file: h:\BYkaThesis_finalCodes\findRefNode_script_recurr_v2.m
%=========================================================================
%
%  KEYING PARAMETERS
%
clear;
tStart = datestr(now);
m = 24;
pr = 31;
eta = 5;
N = 4;
q = 65521;%2^18 -5;% 2^17-1; 2^19-1; 2^20-3; %65521;
Q = N*eta;
%-------------------------------------------------------------------------
% RUN PARMETERS
runs=1000;
txteta = int2str(eta);
txtN = int2str(N);
txtpr = int2str(pr);
txtruns=int2str(runs);
fName = strcat('results2\findRefNode_script_recurr_v2_results',
    '_p',txtpr,'_e',txteta,'_N',txtN);
% Initialise
foundRefNodesCtr(1)=[0];
capturedTags=[1234];
permCouplings(1)=[0];
for r=1:runs % SHOULD BE MOVED DOWN TO USE SAME SET
% MAKE MASTER KEY MATRICES FOR EACH NEW RUN;
  mset = masterKeys(m,N,pr);
% GET A VALID PUBLIC KEY TAG FOR NodeA
% CLEAR PREVIOUS CAPTURES, COUNTS
  clear capturedTags;
  display('New tags');
  savedCtr = 0;
  couplingsA(1) = 0;
  ctrCoup=1;
  prodCouplings = 1;
  permCtr = 0;
```

```
  validKey=false;
  while ~validKey
    idtagA = floor(rand()*q/eta)*eta;
    validKey = isValidPKeyId(m,pr,q,idtagA);
  end %while
  k=1; % COUNTER
  capturedTags(k)=idtagA; % KEEP TRACK OF THE CAPTURED NODES
%=====================================================================
% LOOP UNTIL FOUND TRAITOR NODE
%---------------------------------------------------------------------
  foundRef = false; % Traitor node
  counter = 0;
  while ~foundRef
    counter=counter+1;
% GET A NEW VALID PUBLIC KEY TAG AND MAKE NODE B
    validKey=false;
    while ~validKey % FIND A VALID KEY TAG
      idtagB = floor(rand()*q/eta)*eta;
      tf = ismember(idtagB,capturedTags);
      if (~tf & isValidPKeyId(m,pr,q,idtagB))
        validKey=true;
      end %if %IF ~tf
    end  % while ~validKey
    bIdset{1}=idtagB; % ID SET FOR NODE B
    for i=2:eta
      bIdset{i}=bIdset{i-1}+1;
    end %for i;
    nodeB = makeNode(mset,bIdset,pr,q);
%---------------------------------------------------------------------\
%  SAVE SOME NODES FOR CHALLENGE
%---------------------------------------------------------------------
    if savedCtr < 201 % NUMBER OF NODES TO SAVE
      savedCtr = savedCtr + 1;
      savedNodes{savedCtr} = nodeB;
    end;
    if savedCtr >200
      save('nodesKeys_N7_e6','savedNodes');
      break;
    end; %if
%---------------------------------------------------------------------
    k=k+1;
    capturedTags(k) = idtagB; %SAVE B's ID INTO CAPTURED SET
% JUST TO SEE SOMETHING
    capNodes = size(capturedTags,2)
    disp('Run no ');disp(r);
    disp('eta ');disp(eta);
    disp('N   ');disp(N);
    disp('pr  ');disp(pr);
%=========================================================
%  NOW TEST NODE B WITH ALL PREVIOUS NODES
%---------------------------------------------------------
%   CREATE EACH PREVIOUS TESTED NODE AS NODE A
    x = 1;
    while ((~foundRef) & (x<=capNodes-1))
      x=x+1;
      aTag = capturedTags(x);
      aIdset{1}=aTag;
      for i=2:eta
        aIdset{i}=aIdset{i-1}+1;
      end% for;
      nodeA = makeNode(mset,aIdset,pr,q);
      %-------------------------------------------------------------------
      %  COMPUTE THE PARTIAL KEY-SET, Ra, Rb
      %  TEST IF PARTIAL KEY-SET Rb HAS <= N COUPLINGS WITH Ra RECURRSIVELY
      i=1;
      foundRef = false;
```

```
      while ((~foundRef )& (i <= eta)) %FOR EACH PUB KEY IN A
        Ra = partialKeySet(nodeA,nodeB.id{i},pr,q);
        j=1;
        while ((~foundRef) & (j <= eta))  %FOR EACH PUB KEY IN B
          Rb = partialKeySet(nodeB,nodeA.id{j},pr,q);
          %  TEST THE COUPLERS
          [couplers,ia,ib] = intersect(Ra,Rb);
          % ================================================================
          % COUNT # OF COUPLINGS FOR HISTOGRAM
          % COMPUTE LIMITED CAPTURE PERMUTATIONS
          % ----------------------------------------------------------------
          if ctrCoup <= 20000 %DON'T WANT TOO MANY
            countCouplings = countMatrix(Ra,couplers);
            couplingsA(ctrCoup) = sum(countCouplings{1});
          end % if ctrCoup
          if ctrCoup <= m
            prodCouplings = prodCouplings*sum(countCouplings{1});
          end
          ctrCoup = ctrCoup+1;
          % ----------------------------------------------------------------
          if size(couplers,2) > N  %TOO MANY, PRESENCE OF FALSE COUPLINGS
            %  noRefNode;
            %  disp('No reference node');
            else %COUNT COUPLINGS IN Ra, AND Rb
            couplA = countMatrix(Ra,couplers);
            %  couplingsA(r) = sum(couplA{1}) % SAVE FOR PLOTTING DISTRIBUTIONS
            couplB = countMatrix(Rb,couplers);
            if (sum(couplA{1}) <=N )
              foundRef = true;
              refNode.prKey = nodeA.prKeys([couplA{2}()]); %couplA{2}() for MATLAB
              refNode.id = nodeA.id{j};
              disp('found TRAITOR A =============');
              disp('Ra :');disp(Ra);
              disp('Rb :');disp(Rb);
              % pause
            elseif (sum(couplB{1}) <=N) % EITHER CONDITION IS VALID
              disp('found TRAITOR B ============ ');
              disp('Ra :');disp(Ra);
              disp('Rb :');disp(Rb);
              foundRef = true;
              refNode.prKey = nodeB.prKeys([couplB{2}()]);
              refNode.id = nodeB.id{i};
              %pause
            end  %if sum(couplA)
          end  %if size(couplers)
        j=j+1;
        end% while  ~ foundRef
      i=i+1;
      end%while % foundRef
    end% while % x<=
  end%while % while foundRef
  %disp('found traitor');
  disp('pr ');disp(pr);
  disp('eta');disp(eta);
  disp('N ');disp(N);
  disp('Total runs ');disp(runs);
  disp('run no ');disp(r);
  foundRefNodesCtr(r)=counter
  permCouplings(r) = prodCouplings
  disp('=====================================================')
  %pause
end % for runs
capsize=size(capturedTags,2)
tStart
tEnd=datestr(now)
save(fName,'couplingsA','foundRefNodesCtr','permCouplings','tStart','tEnd','savedNodes');
```

```
%=========================================================================
% COMPUTES THE PARTIAL KEY-SET
% file: partialkeySet.m
%=========================================================================
function [Ra] = partialKeySet(nodeA,idB,p,q)
% k = keyPairs(nodeA,idB,m,p,q)
% returns a (1 x n2N) row vector of secret numbers
  nN=size((nodeA.prKeys),2);
  m = size((nodeA.prKeys{1}),2);
  pubKey=publicKey(idB,m,p,q);
  for j=1:nN
  Ra(j)=mod(nodeA.prKeys{j}*pubKey,p);
end
% kpair=sort(sNumb);
end

%=========================================================================
% file: fullKeySet.m
%=========================================================================
function [keyRX] = fullKeySet(nodeX,nodeY,p,q)
% k = keyPairs(nodeA,idB,m,p,q)
% returns a (1 x n2N) row vector of secret numbers
  nN=size((nodeX.prKeys),2);
  m = size((nodeX.prKeys{1}),2);
  eta = size((nodeX.id),2);
  k=0;
  for i=1:eta
    idY = nodeY.id{i};
    pubKeyY=publicKey(idY,m,p,q);
    for j=1:nN
      k=k+1;
      keyRx(k)=mod(nodeX.prKeys{j}*pubKeyY,p);
    end
  end %for i
  keyRX = sort(keyRx);
% kpair=sort(sNumb);
End

%============================================================
% RETURNS THE PUBLIC COLUMN VECTOR
% publicKey.m
%============================================================
function puKey = publicKey (id, m,p, q)
% Compute the public key,
% Usage puKey = publicKey (seed,size,prime)
% m = rows(ma);
  if (isValidPKeyId(m,p,q,id)==1)
    i=1;
    puKey(i,1)=1;
    temp=1;
    for i=2:m
      temp=temp*id;
      while (temp > q) % mod q
        temp=temp-q;
%       disp("key element after mod ");disp(temp);
      end;
      puKey(i,1)= mod(temp,q);
    end
  else
% puKey = 0;
  end
end
```

```
%============================================================
% GENERATES N MASTER KEY SYMMETRIC MATRICES
% file: masterKey.m
%============================================================
function mKey_set = masterKeys(m,N,p)
% Generates the set of master keys
% Usage mKey = masterKey(size,N,prime)
  for k=1:N
    for i = 1:m
      for j = 1:m
        mKey(i,j) = floor(p*rand());
        mKey(j,i) = mKey(i,j);
      end
    end
    mKey_set{k} = mKey;
  end
end


%============================================================
% CHECKS IF A PUBLIC KEY SEED IS VALID FOR USE
% file: isValidPKeyId.m
%============================================================
function validPKey = isValidPKeyId(m,pr,q,id)
% check if at least one element is > q, and is not zero congruent pr
  validPKey = false;
  temp=1; it=1;
  for (it=1:m-1) % see if at one element is > q;
    temp=temp*id;
    larger=false;
    while (temp>q)
      temp=temp-q;
      larger=true;
    end;
    if ((mod(temp,pr)~= 0) & larger) % check that one is not zero
      validPKey=true;
    end
    it=it+1;
  end
% disp(temp);
% endif
end

%============================================================
% CREATE A NODE OBJECT WITH IdS AND PRIVATE KEYS
% file: makeNode.m
%============================================================
function node = makeNode(m_set,id_set,p,q)
% creates a node given master keys, the node's id set
% Returns an object with node.id, node.prkeys
% Usage node = makeNode(m_set, id_set, p, q)
  eta = size(id_set,2);
  N = size(m_set,2);
  m = size(m_set{1},1);
  ids = id_set;
  y=0;
  for i=1:eta
    puKey=publicKey(ids{i},m,p,q);
    for j=1:N
      y=y+1;
      prKey_set{y} = mod(puKey'*m_set{j},p);
    end
  end
  node.id=ids;
  node.prKeys=prKey_set;
end
```

B.3 Estimating Traitor Node Capture Size and Φ

Genius math script

```
#=========================================================
# CALCULATES THE PROBABILITY OF FINDING A TRAITOR NODE
# CALCULATES MOST OTHER QUANTITIES
#=========================================================
function keySizePilot() =(
  mk=[12,16,24,32];
  p=[7,13,19, 31,37,41,43,47,53,59, 61, 19,23,31,41,251,1023];
# secLevel = 24.08; #28.90; #33.72; #19.2;
  keys = [64,80,96,112,128,192];#; #96;# 64;
# secLevel = log10(2^keysize);
  for ik = 6 to 6  do (
    keysize = keys@(1,ik);
    secLevel = log10(2^keysize);
    bestMem = 2000;
    bestTime = 1000;
    capmin = 10000;
#   for im = 2 to 4 do (
    for msize = 24 to 24 do (
      for ip = 2 to  4 do(
        for ei= 2 to 6 do (
          for Ni = 2 to 6 do (
            # msize = mk@(1,im);
            pr = p@(1,ip);
            Nn = Ni*ei;
            Nn2 = Nn*ei; #eta@(1,ie);
            Na = Nn-Ni;
#================================================
# KEYSPACES
#================================================
           kspace = (Nn2+pr-1)!/(pr-1)!/(Nn2)!;
#          display(" keyspace :",kspace);
           ksbits = log2(kspace);
#          display(" ks bits ",ksbits);
#=======================================================
# EXACT PROBABILITY OF SET Ra or Rb DISJOINT WITH Rc
#  OR Ra, Rb AND Rc ARE DISJOINT
#  OR Ra DISJOINT WITH (Rb U Rc) OR VICE VERSA
#=======================================================
# PRECALCULATES Qa
           Qa@(1,1) = 1;
           for i=2 to Na do (
             sum2 = Qa@(1,1);
             sumQ = 0;
             for j=1 to i-1 do (
               sumQ = sumQ+i!/(i-j)!/j!*Qa@(1,j);
             ); #endfor
             Qa@(1,i) = i^Na-sumQ;
           ); #endfor
#%-------------------------------------------------------------------
# PRECALCULATES Qrc
           Qc@(1,1) = 1;
           for i=2 to Ni do (
             sum2 = Qc@(1,1);
             sumQ = 0;
             for j=1 to i-1 do (
               sumQ = sumQ+i!/(i-j)!/j!*Qc@(1,j);
             ); #endfor
```

```
            Qc@(1,i) = i^Ni-sumQ;
          ); #endfor
#%-------------------------------------------------------------------
#% FOR CASE SET A, B AND C ARE ALLL DISJOINT
#% PROBABILITIES FOR SINGLE, DOUBLE, ... Na INTEGERS IN Ra
          tempP = 0;
          perm3sets = 0;
          for r = 1 to Ni do (
          sumP = 0;
          for k = 1 to Na do (
            if ((pr-k)>=0 ) then  (
              myCa = pr-r;
              for j=1 to k-1 do (
                myCa = myCa*(pr-r-j);
              );
              myCa = myCa/k!;
              sumP = sumP+myCa*Qa@(1,k)*(pr-r-k)^Na; #% only (p-rc-r) left for B
            );#endif
          ); #endfor
          myCc = pr; #% COMBINATIONS WITH LARGE NUMBERS
          for j=1 to (r-1) do (
            myCc=myCc*(pr-j);
          );#endfor
          myCc = myCc/r!;
          tempP = myCc*Qc@(1,r);
          perm3sets=perm3sets+tempP*sumP;
        ); #endfor
#% FOR CASE SET A IS DISJOINT WITH B UNION C
        perm2sets = 0;
        for r = 1 to Na do (
          if ((pr-r) >= 0) then (
            myC2 = pr; #% COMBINATION WITH LARGE NUMBERS
            for j=1 to r-1 do (
              myC2=myC2*(pr-j);
            );#endfor
          myC2 = myC2/r!;
          perm2sets = perm2sets+myC2*Qa@(1,r)*(pr-r)^Nn;
        );#endif
      );#endfor %r
#%-------------------------------------------------------------------
#% COLLATE THE RESULTS
      probab1 = (2*perm2sets-perm3sets)/(pr^(Nn+Na));
#     display("====================","");
#     display("p ",pr);
#     display("e",ei);
#     display("N",Ni);
      display("Pt",probab1);
# ================================================
# CAPTURE SIZES
# ================================================
      cap = 1/2*(1+sqrt(1+8/(ei*ei*probab1)));
# ================================================
# LIMITED CAPTURE BINOMIAL APPROXIMATION
# updated June 27 using binary search
# ================================================
      Cx = Nn!/(Nn-Ni)!/Ni!;
      X = (probab1/Cx);
      stP = 0.7;
      enP = 0.9;
      pbt = stP;
      goOn = true;
      while goOn do (
        pb = pbt;
        y=pb^Ni*(1-pb)^(Nn-Ni)-X;
        if (abs(y)>1e-20) then (
          if (y>0) then (
```

```
            delta=(enP-stP)/2;
            stP = stP+delta;
            pbt = stP;
          )
        else (
          delta=(enP-stP)/2;
          stP = stP-delta;
          enP = stP+delta;
          pbt = stP;
        ); #endelse
      ) #endif abs
      else (
        goOn = false;
      );
    ); # while
    display("pb         ",pb);
    Expect = (Nn*pb);
    display("expected ",Expect);
    limitedCPermE = (Expect)^msize;
    display("Phi", limitedCPermE);
# ================================================
# ROM, RAM AND TIME
# ================================================
    tcomp = 0.0428*(msize*Nn2+(msize-2)*ei)+23.72;
    prkeysize = Nn*msize;
    pairKeysize = Nn2;
#    bestMem = 1000;
#    bestTime = 1000;
#    secLevel = 19.2;
#    capmin = 10000;
#    keysize = 64;
#    if (ksbits >= keysize ) then (
#      if (log10(limitedCPermE)>=secLevel) then (
#        if ( cap >= capmin)  then (
          if (tcomp<bestTime) then (
#           if (prkeysize<bestMem) then (
              bestTime = tcomp;
              bestMem = prkeysize;
              bestN = Ni;
              besteta = ei;
              bestm = msize;
              bestpr = pr;
              bestNc = cap;
              bestSec = log10(limitedCPermE);
              bestPrKeySize = prkeysize;
              bestKpairsize = ksbits;
#           ); # endif prkeysize
          ); # endfor N
        ); #endfor; #e
      ); # endfor; #p
    ); #; endfor; #m
  display("==========","  new ");
  display("Key size  ",keysize);
  display("Keypair   ",bestKpairsize);
  display("cap       ",bestNc);
  display("Sec level ",bestSec);
  display("ROM       ",bestPrKeySize);
  display("Best time ",bestTime);
  display("pr        ",bestpr);
  display("m         ",bestm);
  display("eta       ",besteta);
  display("N         ",bestN);
  ); # for keysize;
);
```

B.4 TinyOS Code for the MICAz mote

Program File: BYkaP.nc

```
// ***************************************************************************
// MODULE FOR GENERATING BYka KEY
// USEAGE: genKpair(uint8_t *ptrKey, uint16_t IdB)
// INPUT:  IdB PUBLIC KEY ID FOR NEIGHBOUR
// OUTPUT: ptrKey is pointer to uint8_t key[16]
// ***************************************************************************
// * ===============  REMOVE FOR SIMULATION ===================================
#include <avr/pgmspace.h>  // REMOVE FOR TOSSIM
module BYkaP{
  provides interface BYka;
}

implementation{
  uint32_t vs, vsTemp; // public key vector seed
  uint16_t idx; // index for private key elements, Nnm
  uint32_t sNTemp=0;      // temp key-set number, largest is 30x30, 10 bits
  uint32_t sNumb[Nnn]; // the key-set numbers, largest mx30x30, 24 bits
  uint16_t s; // index for sNumb, largest is N*n*n
  uint8_t i,j; // counters for N, n
  uint8_t k; // counter, largest Nn,
  uint32_t pubKeyE; // temp public key element, largest 17 bits
  uint8_t prKeyTemp; // temp private key element read from FLASH
  uint8_t *tempPtr; // pointer to OUTPUT key array
  uint8_t BIN[pr]; // key-set occurrences

//  These are in prKey*.h FILE
//  N = number of master keys
//  n = number of public keys
//  m = master key matrix size
// ============================================================================
// INITIALISE Key-set WITH 1st elements of PrivateKey[0]
// TO SAVE ON COMPUTATION SINCE PubKey[0] is always 1.
// ============================================================================
command void BYka.genKpair(uint8_t* ptrKey, uint16_t IdB){
    for (i=0;i<n;i++){
        s = i*Nn;
        for (k=0;k<N*n;k++){           // k is private key index
            idx = k*m;                     // points to 1st in each Private key
// * ---------------- FOR TOSSIM ONLY -----------------------------
//          sNumb[s] = prKey[idx];           // FOR TOSSIM ONLY
// * ---------------- FOR MICAZ ONLY ------------------------------
            sNumb[s] = pgm_read_byte(&prKey[idx]);  // READ FROM FLASH
// ----------------------------------------------------,
            s++;
       }
   }
// ============================================================================
// MAIN BODY OF BYka PROTOCOL -- COMPUTE THE BYka PAIRWISE Key-set
// ============================================================================
    vsTemp = IdB;                                  // IdB is ID of neighbour
    for(i=0;i<n;i++){                              // for each public key
        pubKeyE = 1;                               // PubKey[0] is always 1
        vsTemp = vsTemp+i;                         // increment the neighbour's ID
        for (j=1;j<m;j++){                         // start at 2nd element of pubKey
            pubKeyE = (pubKeyE*vsTemp) % q;        // Public key element
            for (k=0;k<Nn;k++){                    // k is private key index
                idx = k*m+j;                       // points to element in PrivateKey
                s = i*Nn+k;                        // points to Key-set element
// * ------------------FOR TOSSIM ONLY ------------
//              sNTemp = (prKey[idx]*pubKeyE);     // for TOSSIM only
```

```
// * ------------------FOR MICAZ ------------------
                prKeyTemp = pgm_read_byte(&prKey[idx]);
                sNTemp = prKeyTemp*pubKeyE;        // for MICAz
// ----------------------------------
                if (j==m-1){                       // only do modulo at last
                    sNumb[s] = (sNumb[s] + sNTemp) % pr;
                }
                else {
                    sNumb[s] = (sNumb[s] + sNTemp);
                }
            }
        }
    }
  //-------------------------------------------------------------------------
  // SORT Key-set INTO BINS
  //-------------------------------------------------------------------------
  for (i=0;i < pr; i++){ BIN[i] = 0;} // Initialise BIN to zeros
  for (s=0;s < Nnn; s++){
    BIN[sNumb[s]]=BIN[sNumb[s]]+1;
  };
// ================= End of BYka pairwise Key-set computations  ================
//
// ========== MAKE THE FINAL BYka KEY ========================================
// * This method fills and ADD each BYka key with corresponding elements
// * BIN values mod 255,
for (i=0; i<keySize; i++){  // INITIALISE KEY VALUES TO ZERO
ptrKey[i]=0;
}

i = 0;
for (j=0; j<pr; j++){
ptrKey[i]=(ptrKey[i]+BIN[j]) % 255;  // keep size < 256
i = (i+1) % (keySize-1);
}

// ============================================================================
// FOR TOSSIM SIMULATION ONLY -> SHOW BYka KEY-SET AND BIN
for (s=0;s<Nnn;s++){
dbg("keySet","%u \n",sNumb[s]);
}
for (k=0;k<pr;k++){
dbg("BIN","in BIN at %u is  %u \n", k, BIN[k]);
}
}
// ----------------------------------------------------------------------------
} //End command genKPair() ===================================================
```

Configuration File: BYkaC.nc

```
configuration BYkaC{
}
implementation {
  components MainC,LedsC,BYkaP;
  components new TimerMilliC() as uTimer0;
  BYkaP.Boot -> MainC.Boot;
  BYkaP.Leds -> LedsC;
}
```

TinyOS Interface File: BYka.nc

```
interface BYka{
  command void genKpair(uint8_t* ptrKey, uint16_t IdB);
}
```