
Multiparty security on low-power mobile devices

Tobias Limmer

angefertigt an der

Griffith University
Gold Coast Campus, Australia
Faculty of Engineering and Information Technology
School of Information Technology

Studienarbeit im Fach Informatik

Institut für Informatik
Lehrstuhl für Rechnernetze und Kommunikationssysteme (Informatik 7)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Dr. Ruben Gonzalez und
Dr.-Ing. Falko Dressler
Betreuender Hochschullehrer: Prof. Dr.-Ing. Reinhard German

Dezember 2005

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe. Die Arbeit wurde als Abschlußarbeit des Studiengangs 'Bachelor of Information Technology with Honours' an der Griffith University, Australien eingereicht und wurde keiner weiteren Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt und von dieser als Teil einer Prüfungsleistung angenommen. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

I declare that the work is entirely my own and was produced without assistance from third parties. The work was submitted as dissertation in the program 'Bachelor of Information Technology with Honours' at Griffith University, Australia and it was never presented in an identical or slightly adapted version to any other examination board which accepted it as part of a degree. All references, direct or indirect, are indicated as such and have been cited accordingly.

Erlangen, den 30.12.2005

Tobias Limmer

Abstract

Mobile computing is spreading rapidly through the use of low power mobile phones. An increasing number of applications are developed for these devices which exchange data with other devices. One important consideration is data security especially during wireless data transfer in this scenario. This is particularly important in secure groupware, collaborative or multiuser applications where simultaneous communication with multiple parties must be securely maintained. Unfortunately SSL as a point-to-point protocol is suboptimal and mobile phones place serious limitations on the use of cryptography in this scenario, because those devices offer very limited processing resources..

Secure group protocols solve this problem by distributing identical keys to all group members, so that no message has to be encrypted twice. Their two most important parts are the key exchange protocol and its authentication method. This thesis will evaluate important aspects of key exchange protocols and determine bottlenecks in their execution. The calculation of operations in asymmetric cryptographic algorithms and message transfers in the wireless networks cause delays in the execution. So several asymmetric algorithms are tested in the environment of mobile phones. Based on those results, authenticated multiparty key agreement protocols with different designs are theoretically analyzed and compared with each other. The results are compared with a real world benchmark to determine their credibility.

Acknowledgement

I am very grateful to all those who helped me to complete this thesis. First and foremost I want to thank my supervisor Dr. Ruben Gonzales for his extremely helpful guidance and his valuable comments. He helped me to extend my knowledge in various areas and encouraged me to think critically. It was an enriching experience to be introduced into the world of scientific research by him.

Furthermore, I would like to thank my supervisor Dr. Falko Dressler in Germany and Dr. Renate Sitte for guidance.

I would also like to give special thanks to my friends Corinna and Shaun for their valuable comments and to my family for their encouragement.

The staff at the School of Information Technology was also very helpful by providing a good working environment and contributing their time and effort.

Contents

List of Figures	iv
List of Tables	v
Acronyms	vii
1 Introduction	1
1.1 Thesis Structure	3
1.2 Hypothesis	3
2 Overview and related Work	4
2.1 Mobile phones	4
2.1.1 Hardware	4
2.1.2 Java	5
2.1.3 Mobile networks	7
2.1.4 Protocols in use	8
2.2 Cryptographic algorithms	9
2.2.1 Symmetric algorithms	9
2.2.2 Asymmetric algorithms	10
2.3 Communication	15
2.3.1 Cryptographic protocols	16
2.3.2 Key exchange	18
2.3.3 2-party key exchange protocols	19
2.3.4 Multiparty key exchange protocols	20
2.3.5 PKI	24
3 Secure group key exchange	26
3.1 Analysis of secure protocols	26
3.1.1 Considerations	26
3.1.2 Assumed conditions	30
3.2 Asymmetric cryptographic algorithms	31
3.2.1 Overview	31

3.2.2	Benchmark	36
3.3	Comparison of key exchange protocols	36
3.3.1	Contributory key agreement	38
3.3.2	Centralized key distribution	43
3.3.3	Benchmark	49
4	Results and Discussion	50
4.1	Asymmetric cryptographic algorithms	50
4.1.1	RSA	51
4.1.2	ECC	54
4.1.3	XTR	55
4.1.4	Discussion	55
4.2	Comparison of key exchange protocols	57
4.2.1	Contributory key agreement	58
4.2.2	Centralized key agreement	61
4.2.3	Comparison	62
4.2.4	Benchmark	64
5	Conclusion and Future Work	67

List of Figures

2.1	The Nokia 6610 mobile phone	5
2.2	Overview of the Java 2 platform with all its flavours for different hardware configurations (©Sun Microsystems)	6
2.3	Architecture view of the Java platform inside a mobile device (©Sun Microsystems)	7
2.4	Graphical representation of the elliptic curve $y^2 = x^2 - 7x$	12
2.5	Choice of network topologies for group communication	16
2.6	process of 2-party key exchange and rekeying	17
2.7	different types of multiparty key exchange	21
2.8	structure of TGDH by Kim [32]	23
3.1	sample key structure of TGDH	41
3.2	Initial setup of the TGDH protocol for 4 users	42
3.3	Process of key distribution with a group manager	44
3.4	The LLK protocol	46
3.5	Analysis of LLK in an initial multiparty key exchange	47
4.1	test	51
4.2	Encryption benchmark results in series 40 mobile phones	52
4.3	Decryption benchmark results in series 40 mobile phones	52
4.4	Key generation benchmark results in series 60 mobile phones	53
4.5	Encryption benchmark results in series 60 mobile phones	53
4.6	Decryption benchmark results in series 60 mobile phones	54
4.7	Performance comparison of RSA, ECC and XTR on series 40 and 60 mobile phones	55
4.8	Speed of a series 60 mobile phone compared to a series 40 device	56
4.9	Comparison of initial setup for 5 members between series 40 and 60 mobile phones	57
4.10	Performance of protocol AKE1 for initial setup	58
4.11	Performance of protocol AKE1 for member join	59
4.12	Performance of protocol AKE1 for member leave	59

4.13	Performance of protocol TGDH for initial setup, member join and leave	60
4.14	Performance of centralized multiparty protocol using LLK for initial setup	61
4.15	Performance of centralized multiparty protocol using a 2-party key exchange protocol similar to MTI/A0	62
4.16	Performance comparison of multiparty protocols with 5 group members	63
4.17	Results of the centralized key exchange benchmark with 3 mobile phones using LLK	64
4.18	Time diagram of the benchmark's key exchange process for initial setup	65

List of Tables

2.1	speed comparison of RSA and XTR at similar security levels . . .	14
3.1	key sizes of similar security levels for different cryptographic algorithms	35
3.2	Comparison of various ECC 2-party key exchange protocols	45

Acronyms

AKA:	auxiliary key agreement
API:	application programming interface
CA:	certificate authority
CPU:	central processing unit
DH:	Diffie-Hellman
EC:	elliptic curve
ECC:	elliptic curve cryptography
ECDSA:	elliptic curve digital signature algorithm
ECIES:	elliptic curve integrated encryption scheme
GEK:	group encryption key
GM:	group manager
HTTP:	hypertext transport protocol
IKA:	initial key agreement
IP:	internet protocol
KEK:	key encryption key
MAC:	message authentication code
MIDP:	mobile information device profile
OSI:	open systems interconnection
PKI:	public key infrastructure
SSL:	secure socket layer
TLS:	transport layer security
TTP:	trusted third party
UDP:	user datagram protocol

Chapter 1

Introduction

In the last few years mobile computing has experienced an immense rise in popularity, mainly caused by the wide-spread use of mobile phones. For mobile devices one of the most important issues is the security of the exchanged data. The transfer of data itself is prone to become the weak link, as very few encryption methods are available for applications running on mobile phones. It allows third parties to easily intercept the data during wireless transfer and during the routing through an insecure network to its destination, such as the internet.

This issue is even more important in groupware, collaborative or multiuser applications, where simultaneous communication between different parties must be securely maintained and all the peers are realized on mobile devices. Yuan and Long expected in their article from 2002 [72], that “Peer groups and subscription-based multicast applications will be a major model for the smart wireless applications of the future”. Examples of these types of applications are teleconferencing, video streams or shared white boards in groupware applications. In those cases one peer usually sends identical data to all the other peers. The data is distributed within the group so that all participating members are synchronized. So if someone talks during a teleconferencing session, the recording device transmits the data to all the other peers simultaneously.

The network topology considered for this multiparty communication is a fully-meshed net: every member of the group exchanges data with all the others. Multiparty communication could also be realized in a star formation: there group members only have one connection to a group controller which can be realized with a server. Mobile devices would not be appropriate as group controllers, as this type of group management needs a lot of computational and network related resources. The usage of external devices introduces restrictions on the usability of multiparty protocols, so this solution is avoided in this work.

An important aspect for this communication type is dynamic membership: members should be able to join and leave the group without any restrictions.

One trivial solution would be separately managed point-to-point connections between all the group members and the usage of already well established secure 2-party protocols as e.g. SSL. This solution is clearly suboptimal as it does not take advantage of multiparty group communication's unique features.

Several secure protocols for multiparty applications have already been suggested. They can be roughly separated in contributory protocols and centralized protocols. Contributory protocols treat every party member identically and do not depend on a host which manages the security setup of the group. Centralized protocols make use of one central host called group manager (GM) which controls the group's security. They are slightly less secure than contributory protocols, as the dependence on a single GM may make it easier to exploit the protocol by an adversary.

Authentication is a major aspect of ensuring private secure communication. Protocols without authentication are vulnerable to man-in-the-middle attacks, where an adversary listens to and modifies the transferred networks packets between two communicating hosts and is able to decipher the messages. Authentication can be performed in several ways: using passwords or certificates which include a public key, and the underlying cryptographic algorithms can be symmetrical or asymmetrical.

The main limitation for the usage of cryptography are the mobile phones' extremely limited processing resources. Several performance tests have already been performed for handheld devices. Those devices are several times faster than mobile phones, so it is not known yet, how fast cryptographic algorithms and especially key exchange protocols run on mobile phones. They are much slower, because the emphasis of mobile phones development is on voice communication and long battery runtimes. But their support for applications is improving: The processor speeds are increasing and more and more features are offered in their program environment. There is a tendency for devices to converge with low-end handhelds. But despite this development, their processing power will still be a major concern for encryption in the years to come.

This thesis will investigate multiparty key exchange protocols suitable for mobile phones. We will analyze the asymmetric cryptographic systems RSA, ECC and XTR, as they provide means for authentication and many secure protocols are based on those algorithms. We will determine their performance in our target environment and decide, what system is most suitable for mobile phones. Then secure multiparty protocols will be analyzed and their speed will be theoretically determined using the results of the tested asymmetric cryptographic systems. The following protocols will be used in the analysis:

- AKE1 by Bresson et al. [8]

- TGDH by Kim [32]
- a centralized multiparty protocol based on LLK by Lee et al. [39] and based on MTI/A0 by Matsumoto et al. [42]

To validate the results, one of the secure multiparty protocols was implemented and its performance was compared to our theoretical results.

1.1 Thesis Structure

The thesis is structured in the following way: Chapter 2 presents an introduction and overview of the topic. It includes the limitations of mobile phones, reviews different cryptographic algorithms and presents several secure communication protocols. Chapter 3 describes used methodology to achieve our goal: In the first section general features of secure protocols are analyzed, then different asymmetric cryptographic algorithms are discussed, and in the last section the speed of key exchange protocols is theoretically calculated. Chapter 4 presents and discusses the results of chapter 3. Chapter 5 summarizes the experimental results of the thesis and proposes further work in this area.

1.2 Hypothesis

There is currently no good solution for secure mobile multiparty communication. If we investigate group protocols and assess their performance on the target platform, we will be able to determine the feasibility of secure multiparty communication on mobile devices and the protocol which offers appropriate security and performs best.

Chapter 2

Overview and related Work

This chapter presents an introduction to the topic of this thesis. In the first section we consider the environment and limitations of mobile phones and a short introduction to secure protocols which are in practical use today. Secure communication protocols are based on cryptographic algorithms, so several algorithms that are used in this thesis are reviewed in the second section. The third section introduces cryptographic protocols and key exchange protocols, which are a component of secure protocols. It also presents several proposed 2-party and multiparty key exchange protocols.

2.1 Mobile phones

2.1.1 Hardware

The processing capabilities of mobile phones are very limited as those devices are focused on providing long battery runtimes and small proportions. However, much effort is being undertaken to improve their capabilities to accommodate more complex functionality and enable applications to run efficiently on the devices. The current assortment of mobile phones can be roughly divided into two classes by regarding the processing power and operating system: feature phones and smart phones. Examples of these types are the Nokia series 40 and series 60 phones. We exclude PDA-like phones in this work.

The term ‘series 40’ was introduced by Nokia. The operating system on those devices is written in Java and their 32 bit ARM-9 CPU has approximately 20 MHz. Compared to a standard desktop computer, this CPU is 500 times slower. A typical representative of a series 40 phone would be the Nokia 6610 (see figure 2.1). Its total memory is 200 KBytes, but for Java applications only 70 KBytes are available. This poses considerable limitations on the programs which run on this device. The figures were taken from [68] and the performance was compared to a test run on a desktop computer.



Figure 2.1: The Nokia 6610 mobile phone

The faster generation is the group of series 60 mobile devices. The term ‘series 60’ was introduced by Nokia and was adopted by several other companies. The devices run on a non-Java operating system called SymbianOS [67]. It also supports downloadable Java applications. Their CPU is considerably faster than the processor of series 40 devices with roughly 120 MHz. The available memory also increases substantially to 2 MBytes and more.

Information on exact hardware specification is not available publicly, so the given values may vary from device to device.

2.1.2 Java

Almost all of the currently available mobile phones implement Java as programming environment. Sun Microsystems gives in [66] a brief overview. As Java is offered for very diverse hardware architectures, it is divided into different platforms. Figure 2.2 shows that J2EE (Java 2 Enterprise Edition) and J2SE (Java 2 Standard Edition) are suitable for servers and personal computers. J2ME (Java 2 Platform, Micro Edition) is mostly used for devices with lower computing power including mobile phones. The capabilities of the device and the available APIs in the Java programming environment are specified in a configuration and a profile.

Configurations define a virtual machine and a basic set of class libraries. They specify the basic functionality for a set of devices which have similar features, as e.g. memory and processor footprint or network connectivity. The Connected Limited Device Configuration (CLDC) is used for mobile phones. It is suitable for 16 or 32 bit CPUs and specifies a minimum of 128KB of memory available for the Java virtual machine and its running applications.

Profiles define a set of higher level APIs that further define the device’s specifications, such as the application life cycle model or the graphical user interface. The Mobile Information Device Profile (MIDP) is the profile for mobile phones

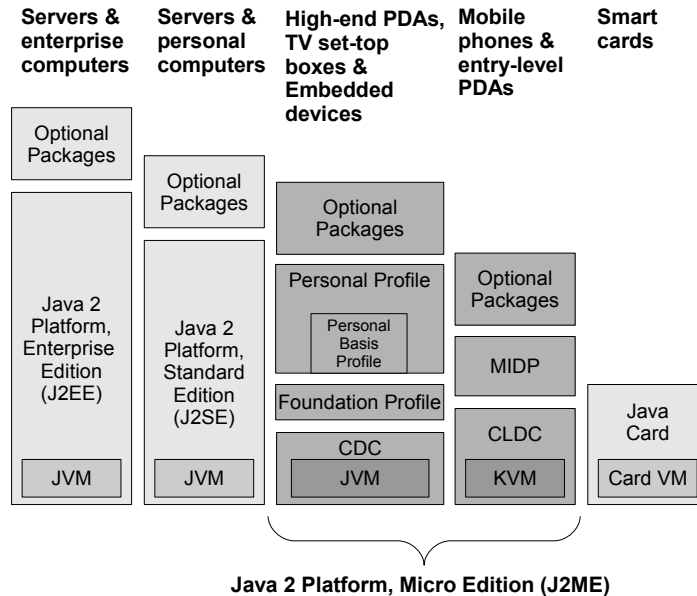


Figure 2.2: Overview of the Java 2 platform with all its flavours for different hardware configurations (©Sun Microsystems)

and PDAs with low processing power. It offers the core functionality for mobile applications and includes the user interface, network connectivity, local data storage and application management.

The current version of MIDP is 2.0 (Specification [30]). Figure 2.3 shows, how MIDP Java applications rely partly on the MIDP and the CLDC. The MIDP also accesses the CLDC. The CLDC itself provides the virtual machine which calls functions of the native system layer. All Java code is interpreted by the Java virtual machine and direct access to the hardware is therefore not possible. This creates a safe environment for the potentially unsafe executed code, as e.g. no buffer overflows are possible there. The virtual machine also offers an internal garbage collector to free unused memory. All Java code is portable between the devices where the same set of APIs is offered, which are specified by the CLDC and the MIDP 2.0. All Java code is run on top of the mobile's real-time operating system whose primary task is to manage network-related functions to ensure connectivity. This task has priority over all other functions and uses up much of the limited processing power of the devices.

Java applications for MIDP are called MIDlets. MIDP 2.0 includes over-the-air provisioning (OTA). This is a specification of protocols and processes which enable mobile phones to download and install MIDlets in a general way. That method is the normal way of installing applications on mobile phones.

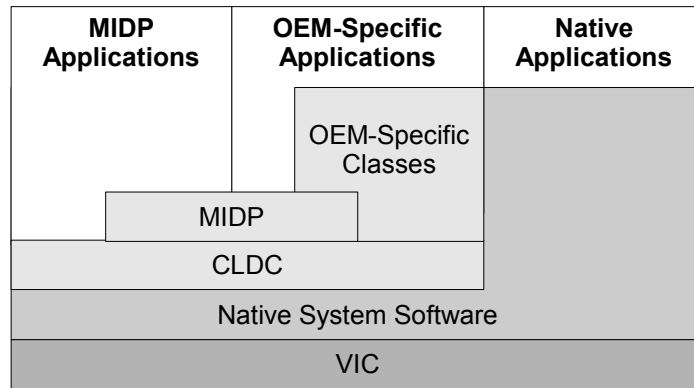


Figure 2.3: Architecture view of the Java platform inside a mobile device (©Sun Microsystems)

MIDP 2.0 also provides HTTPS in a stripped-down version for secure data downloads. The included version of SSL is not available for direct use by applications. It can only be used for a HTTPS connection to an external server, so it can not be used for multiparty groupware applications.

2.1.3 Mobile networks

The performance of data transfers between mobile phones is greatly affected by the mobile networks that the phones use. At the moment there are three major networks available for data transfer: GSM, GRPS and UMTS.

The Global System for Mobile Communications (GSM) is the most popular network used in the world. It is a second generation network and its specifications were completed in 1990. The system's main focus is on voice transmission, so the maximum possible data transfer rate is only 14.4 kbit/s. Data transfer is billed on the time spent on being connected to the internet.

General Packet Radio Service (GPRS) was added to the GSM standard in 1997. It enables mobile phones to make use of unused bandwidth for data communication. Its theoretical speed limit is 170 kbit/s, but in practice usually 30-70 kbit/s are achieved. The cost for this service is based on the amount of transferred data.

The successor of GSM is the third generation Universal Mobile Telecommunication System (UMTS). It was released in 2000 and meets the requirements for higher speeds for data transfer. Its theoretical maximum is 1920 kbit/s, but at the moment most networks only support up to 320 kbit/s.

2.1.4 Protocols in use

This section describes two of the most used secure protocols in the internet. They only support secure end-to-end communication between two entities. Both protocols are not designed to run on restricted environments like those of mobile phones, so we only present a short overview.

SSL/TLS Secure Sockets Layer (SSL) was initially released by Netscape as Version 3.0 in 1996 and is described in RFC 2246 [19]. Meanwhile the successor Transport Layer Security (TLS) is available in version 1.0 and gradually replaces SSL. The protocol provides endpoint authentication and security over the internet protocol in OSI layer 6. Only end-to-end communication, that means communication between two hosts is supported. It uses TCP for data transfer and thus lies between OSI layer 4, the transport layer, and OSI layer 7, the application layer.

The basic phases of the protocol consist of

- negotiation of supported algorithms
- asymmetric key exchange and authentication
- symmetric encryption of application data

SSL offers several choices for its used cryptographic algorithms, which the hosts agree upon in the first phase of the protocol. The key exchange can be accomplished with RSA encryption and the Diffie-Hellman key exchange, both with signed data packets for authentication. Furthermore the Digital Signature Algorithm (DSA) and Fortezza, a security system based on a PC card security token, are offered for authentication. SSL also provides a wide range of symmetric encryption algorithms including RC2, RC4, IDEA, DES, 3DES and AES.

Every application layer protocol which relies on TCP/IP for communication can make use of SSL. All data passed from the application to the SSL layer is encrypted and forwarded to the peer, where it is automatically decrypted and passed on to the application. Many protocols use SSL for encryption. The most popular protocol is HTTP which was the initial reason why Netscape developed this protocol: HTTPS denotes the combination of HTTP and SSL and is the most popular application with SSL. A free implementation of SSL is provided by OpenSSL [52].

MIDP 2.0 also provides an implementation of SSL called KSSL (“kilobyte” SSL). It has only client-side functionality and is a stripped down version of SSL 3.0: only RSA with key sizes up to 1024kbit and RC4 are supported as encryption algorithms. The API is mostly implemented in Java, but performance-critical parts are also realized in native platform code. Gupta and Gupta proposed KSSL

in 2001 [21] and measured speeds on a PalmVx handheld with 20 MHz. RSA verifying operations took 1.4 seconds and RSA signing operations took 80.9 seconds.

IPsec IPsec is a standard to provide security for the Internet Protocol (IP) suite and is defined in the RFCs 2401-2412 [57]. It encrypts and authenticates packets in the network layer (OSI layer 3). There it secures all transferred data between two endpoints and is not restricted to certain applications which have support for it (in contrast to SSL).

IPsec consists of two protocols: Authentication and message integrity, but not data confidentiality, is realized in the Authentication Header (AH). Encapsulating Security Payload (ESP) provides authentication, data security and message integrity. The key exchange is performed by the Internet Key Exchange (IKE) protocol.

2.2 Cryptographic algorithms

A cryptographic algorithm is a function which encrypts or decrypts data. The process of encrypting data is represented as a function E . It takes the plaintext P as input argument and results in the ciphertext C . The decryption process D is used in a similar way. Usually a key K is needed for encryption or decryption. The whole process of encrypting and decrypting data can be represented as

$$E_{K_1}(P) = C$$

$$D_{K_2}(C) = P$$

The size of the used keys K_i directly influence the security of the encryption. The bigger it is, the more difficult is the decryption of the ciphertext without knowing the key. Certain key sizes provide different security levels for different types of cryptographic algorithms.

Cryptographic algorithms can be separated into two groups, depending on the number and type of keys used: Symmetric cryptographic algorithms and asymmetric cryptographic algorithms

2.2.1 Symmetric algorithms

Symmetric algorithms, which are also called private key algorithms, take the same key for both encryption and decryption of data. The equations describing symmetric encryption and decryption of data are

$$E_K(P) = C$$

$$D_K(C) = P$$

If two peers want to transfer symmetrically encrypted data to each other, they need to agree on a common key first. This process is called key exchange. The security of symmetric algorithms relies on the secrecy of the key - if an attacker acquires it, he can decrypt the ciphertext encrypted with it. Symmetric algorithms usually have a good performance, as they are mainly based on the permutation of bits and the XOR-operation. Some of the most used symmetric algorithms are 3DES, AES and Blowfish.

AES is used in this thesis. It was developed by Daemen and Rijmen in 2000 [18]. It is a symmetrical cipher algorithm which is easy to implement and requires minimal computing power and memory. Its key size must be a multiple of 32 bits. 128, 196 and 256 bits are in common use through the AES specification by the US government in 2001.

A message authentication code (MAC) algorithm uses encryption to achieve data integrity and data origin authentication for a message. Methods used to achieve this goal are supposed to be simple and fast, so many implementations use symmetric encryption and hash functions. The following example shows its application: If host U_A wanted to send a message to host U_B and both hosts share a common key k , U_A would encrypt a plaintext message P using a symmetric encryption function E : $C = E_k(P)$. The authentication tag $T = \text{MAC}_k(P)$ would also be computed using the shared key k and the plaintext P . The MAC algorithm hashes P to a small value and then the value is symmetrically encrypted using k . The result is stored in the authentication tag T . The ciphertext C and authentication tag T is transmitted to U_B . U_B then decrypts the ciphertext and checks the authenticity of the plaintext P by applying the same MAC algorithm as U_A . If the resulting value equals T , the message authenticity is verified.

2.2.2 Asymmetric algorithms

Asymmetric algorithms, also known as public-key algorithms, got their name because of their use of a keypair for encryption and decryption. This keypair is generated in an initialization phase and consists of the so called public (W) and private key (w). These keys are related: data encrypted with one key can only be decrypted with the other key of the keypair. The following equations show the process:

$$\begin{aligned} E_w(P) = C_1 & & E_W(P) = C_2 \\ D_W(C_1) = P & & D_w(C_2) = P \end{aligned}$$

The public key is usually made public and everyone can access it. The private key is kept secret. There are two common usage schemes for asymmetric algorithms:

- Person A wants to send encrypted data to person B. Therefore A encrypts data with B's public key (W_B). This ciphertext can only be decrypted with B's private key (W_B). Public key algorithms are slow compared to symmetric algorithms, so data is usually encrypted with a symmetric algorithm and the used secret key for this encryption is encrypted and decrypted by an asymmetric algorithm.
- Person A wants to send digitally signed data to B. To do that, A encrypts data with A's private key (w_A). Everybody else who has A's public key (W_A) can decrypt the data, and as only A could have encrypted data which can be decrypted with A's public key, is sure that indeed A encrypted it. As in the previous case, usually not all the data is encrypted, but a hash function is applied to it and only the resulting hash value is used as input to the asymmetric operations.

The keys in the keypair are mathematically related, so it is possible to calculate the private key from a public key. Therefore this calculation must be as computationally intensive as possible, so that the calculation can not be finished in reasonable time.

RSA

Rivest, Shamir and Adleman presented the first asymmetric cryptographic algorithm in 1978 [58]. It is one of the most used algorithms at the moment. RSA Laboratories has released the cryptographic standard PKCS #1 [59] which describes and defines implementation issues for RSA cryptography to ensure an implementation without security flaws.

Attacks on the RSA public-key algorithm are based on the *discrete logarithm problem*. If we have given the values x , a and n of equation $x = a^b \pmod n$, the discrete logarithm problem describes the problem of finding value b . This problem is computationally very expensive and until now values with a length of 1024 bit are not broken.

In 2000 Boneh et al. suggested a method to speed up RSA key generation on a handheld using an untrusted server. The PalmV handheld contained a 68k processor running at 20 MHz. They created unbalanced RSA keys, which are slightly less secure than a balanced keypair. In their experiments they used a small handheld device to generate the keys and the device needed 7.5 minutes without any speed improvements. As the key generation needs a lot of calculations, some of those are offloaded to one or two servers in their proposal. The servers are untrusted, and they can not derive keys, which were generated on the handheld, from their calculations. The only condition for the method using two servers is,

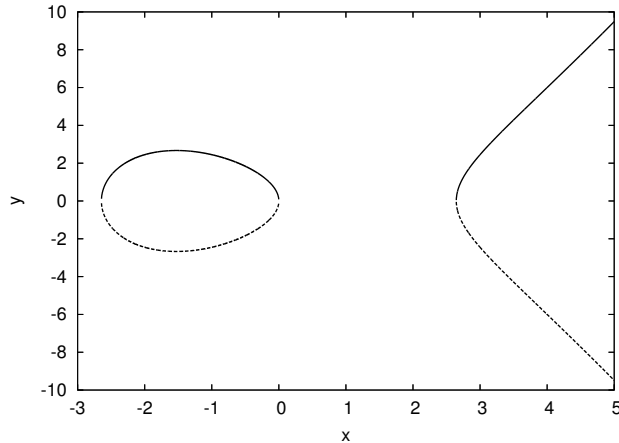


Figure 2.4: Graphical representation of the elliptic curve $y^2 = x^2 - 7x$

that the servers must not be able to exchange information with each other, or else the generated key pair can be recovered. Their experiments showed, that key generation can be sped up to 2.7 minutes with one server and 1.1 minutes with two servers. The authors presented that it is possible to speed up RSA on low power devices, but to do so external servers are needed which may not be available at all times.

Lei et al. proposed another method to outsource computations of asymmetric algorithms in 2004 [40]. They described a digital signature scheme for mobile devices called Server Based Signature (SBS) which consumed less computational resources on the client side. They did not provide experimental results, so it is not comparable to other similar algorithms.

ECC

A more efficient asymmetric algorithm is the elliptic curve algorithm proposed independently by Miller in 1985 [46] and Koblitz in 1987 [34]. Several patents for ECC are owned by Certicom, so it is not freely available. In the beginning the involved calculations were not efficient, but a lot of work has been put into this problem and by the late 1990s they were as fast as RSA. ECC needs smaller key sizes for similar security to RSA, so the time needed for calculations is shortened.

Elliptic curve cryptography is based on mathematics of finite field elliptic curves. An elliptic curve is represented by a simplified Weierstrass equation of the form

$$y^2 = x^3 + ax + b, \quad 4a^3 + 27b^2 \neq 0 \quad (2.1)$$

A graphical representation of an example can be seen in figure 2.4. \mathbb{F}_q denotes the finite field which contains q elements. q is a prime power and determines

the underlying field of the elliptic curve. It can either be a prime field ($q = p$), called ECP, or a polynomial field ($q = 2^m$), called EC2N. Polynomial fields are faster in hardware implementations, whereas prime fields are more efficient in software implementations. A finite field elliptic curve is defined by the set of points $P = (x, y)$ which satisfy equation 2.1. The variables a and b must be chosen carefully, as some combinations enable cryptanalysts easier attacks.

Addition of two points inside the finite field elliptic curve $Q = P + R$ is defined according to a chord-and-tangent rule. Multiplication

$$Q = xP \tag{2.2}$$

is defined as repeated addition of a point P to itself x times. It is sometimes also referred to as exponentiation, although this is technically not correct. It is the main operation of elliptic curves. The elliptic curve discrete logarithm problem is based on the fact, that it is intractable to determine the value x of equation 2.2, if Q and P are given. It might remain unanswerable even if RSA's discrete logarithmic problem of factoring large numbers, can be solved efficiently.

Domain parameters of an elliptic curve define the exact curve which is used for the calculations. It must be known to all entities which want to perform compatible calculations. The domain parameters consist of the

- field size q
- description of the field representation (prime field / polynomial field)
- two field elements a and b which define the elliptic curve equation 2.1
- so-called base point $P = (x_P, y_P)$ on the elliptic curve
- order n of P
- cofactor $h = \#\mathbb{F}_q/n$ (which is usually 1 for prime fields)

To ensure security, those domain parameters have to be checked for consistency.

Elliptic curve theory is completely different to RSA, but operations in that field can be mapped to operations comparable to RSA. As a result, ECC can be used as a replacement to RSA in many cryptographic protocols, although in some cases the converted protocols may become insecure.

ECDSA The Elliptic Curve Digital Signature Algorithm (ECDSA) is the EC version of the Digital Signature Algorithm (DSA). It is the most popular standardized EC-based signature scheme. Signature generation of a message m works in the following way (domain parameters are assumed to be given):

algorithm	key size	key generation	encryption	decryption
RSA	1020 bit	1224 ms	5 ms	40 ms
XTR	170 bit	73 ms	23 ms	11 ms

Table 2.1: speed comparison of RSA and XTR at similar security levels

1. Select k randomly in the range $[1, n - 1]$
2. Compute $Z = kP$
3. Compute $r = x_Z \pmod n$. x_Z is the x-coordinate of point Z . If $r = 0$, then go to step 1.
4. Compute $e = H(m)$
5. Compute $s = k^{-1}(e + wr) \pmod n$. If $s = 0$, then go to step 1.

w is the private key of the host which generates the signature. r and s are added to the message, so that the signature can be verified. H is a hashing function which generates a hash value with the required size.

Signature verification expects the parameters r and s from the generation algorithm as well as the message m and the originating host's public key W . The following steps describe the operation:

1. Compute $e = H(m)$.
2. Compute $w = s^{-1} \pmod n$.
3. Compute $u_1 = ew \pmod n$ and $u_2 = rw \pmod n$.
4. Compute $Z = u_1P + u_2Q$.
5. If $Z = \infty$, then reject signature.
6. Compute $v = x_Z \pmod n$. x_Z is the x-coordinate of point Z .
7. If $v = r$, then accept the signature, else reject it.

The signing operation performs one EC multiplication and the signature verification operation performs two EC multiplications.

XTR

Another interesting alternative to RSA was presented by Lenstra and Verheul in 2000 [41]: XTR. It is an abbreviation for 'Efficient and Compact Subgroup Trace Representation'. Similar to RSA, XTR is based on the discrete logarithm

problem, but it uses traces to represent and calculate powers of elements of a finite field's subgroup. Lenstra owns several patents for XTR. Advantages of XTR are the fast generation of a keypair, small key sizes and fast speed. Its attributes are comparable to ECC. Table 2.1 shows Lenstra's speed comparison. There, XTR is 16 times faster for key generation, 4 times slower for encryption and 4 times faster for decryption compared to RSA. Using XTR or ECC for asymmetric encryption, key generation, the main performance problem of RSA, may be resolved. Unfortunately XTR is not yet thoroughly tested, so real world use may still be risky, as the possibility is higher than in RSA or ECC, that fundamental flaws in its theory are discovered.

Lauter described the advantages of elliptic curve cryptography for wireless security in 2004 [36] and presented an overall introduction to the topic. He compared the key sizes of symmetric, RSA and ECC algorithms. An RSA based cryptographic algorithm with 1024 bit keys is as secure as an EC cryptographic algorithm with a much lower key size of 163 bit. Symmetric ciphers are in comparison more secure, they only need 80 bit keys. Lauter states, that in this case, the exponentiation speed of ECC is 5 to 15 times higher than RSA. As exponentiation is the fundamental operation for ECC, those figures are promising. Those performance advantages are the reason, why many companies adopt the ECC technology at the moment.

Hoffstein et al. proposed another asymmetric algorithm in at Crypto '96 [24]. They described NTRU, a high speed algorithm which creates short keys and has low memory requirements. Its concept is very different from RSA and ECC and has some efficiency advantages. But NTRU has a history of attacks: one version was broken by Coppersmith and Shamir [16] and later other attacks followed. The inventors of NTRU responded with security improvements. No signature scheme is available yet, an initial design and a revised version were both broken. This is why the system has no commercial success and will not be analyzed in this thesis.

2.3 Communication

Many applications emerge nowadays which require not only two-party communication, but also multiparty communication. There, several devices exchange data at the same time with each other. Possible network topologies are (see figure 2.5)

- *Fully meshed.* All devices send information to all other devices inside the group.
- *Star.* One device retains connections to all other devices in the group. Data incoming from devices will be transferred to all remaining devices.

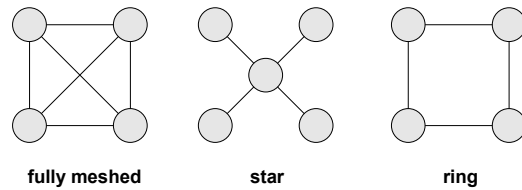


Figure 2.5: Choice of network topologies for group communication

- *Ring*. Every device is connected to two neighboring devices. Data coming from one side will be routed to the other side.

Every topology has its positive and negative aspects, so an appropriate network type has to be chosen depending on the application.

2.3.1 Cryptographic protocols

A protocol is a set of rules which enables two or more computing devices to exchange information. It describes when to send and receive certain data.

Cryptographic protocols can have different purposes: *Authentication protocols* try to establish the authenticity of the communicating peers. No data is encrypted, but all parties are ensured after successful execution of the protocol, that their peers are who they seem to be. *Key establishment protocols* enable communicating parties to agree on a common shared key. This key is needed, as encryption of bulk data is usually performed with fast symmetric algorithms. Bulk data is data which is transmitted by the application that makes use of the secure protocol, e.g. audio and video data transmitted by video conferencing software. This data is encrypted and decrypted by the implementation of a secure protocol and will be called bulk data in this thesis. Symmetric cryptography takes the same key for encryption and decryption of data, so all communicating peers need to use the same key. This process of key establishment is also called key exchange or key agreement. It must not provide insight to the resulting shared key for adversaries who listen to the communication. *Authenticated key establishment protocols* add to the process of key establishment authentication of all peers. Every party can be sure, that after successful completion of the protocol only those devices know the shared key which have been authenticated successfully.

Secure protocols use different types of keys or keypairs:

- *long-lived/static keys*. They are bound to a host for a certain time, this may be several years. They are mainly used for authentication purposes and

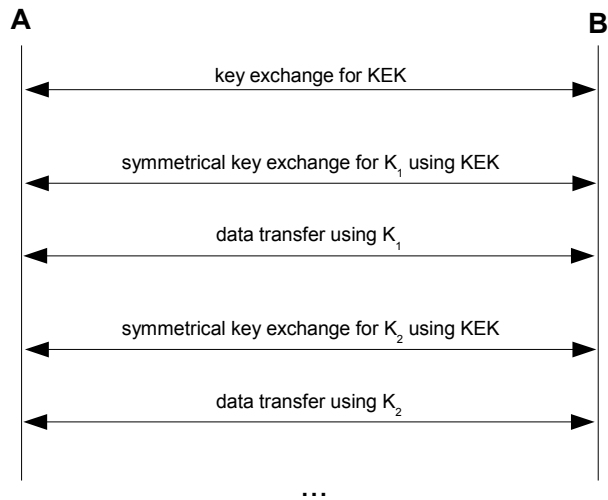


Figure 2.6: process of 2-party key exchange and rekeying

consist of an asymmetric keypair. Usually the public key is stored in form of a certificate.

- *short-lived/ephemeral keys*. Those keys are generated for each protocol run, and last for one session or sometimes only for a part of a session. That is why they are also called *session keys*. As they are used for bulk encryption of data, they are changed regularly. This process limits the available ciphertext for a certain key. If one of the session keys is compromised, only a part of the transferred data can be decrypted. In group communication, all transferred bulk data is encrypted with a common group encryption key (GEK) which is equivalent to the session key in 2-party communication.

It is important to use a common GEK in secure multiparty communication. In a fully meshed network every host transmits data to every other host inside the group. If no common GEK was used, transmitting hosts would have to encrypt data as often as the number of receiving hosts. This is very inefficient, especially if there is a big amount of data to be sent or the group is large. The usage of a common GEK solves this problem: transmitted data has to be encrypted only once and the size of the group does not influence the amount of needed encryptions.

Many secure protocols follow a certain process to establish and maintain secure symmetrical keys. Figure 2.6 shows, that in the beginning a normal key exchange is performed. There a *key encryption key* (KEK) is exchanged. This key is used to exchange short-lived session keys K_i , also called GEKs, which encrypt bulk data for a part of the session. After a certain time both hosts exchange a new session

key K_{i+1} . This process is called *rekeying*.

2.3.2 Key exchange

The most important point for secure protocols is *key exchange*. Usually application data is encrypted with less computationally intensive symmetric algorithms. Those private key algorithms require that parties which want to exchange encrypted data, use the same key for encryption and decryption. The key exchange solves this problem.

Many key exchange protocols have been proposed, but a large amount of them have turned out to be flawed. Designing a secure protocol proves to be a complicated problem.

Key exchange can be performed using symmetric or asymmetric algorithms. Symmetric key exchange protocols are computationally very efficient, but they rely on a predistributed secret key, such as a password. Asymmetric protocols do not necessarily rely on predistributed information. Only if authentication is required, each host that should be authenticated will need to own a long-term secret key pair.

Authenticated key exchange (AKE) ensures, that nobody except the participating hosts can learn any information about the key which they agree on. Thereby we distinguish two different types of authentication:

- *Implicit authentication.* Host A is not assured that host B knows the shared key and that the key exchange was successful. A protocol which provides implicit authentication is also called ‘authenticated key agreement protocol’ (AK)
- *Explicit authentication.* Host A is assured that host B knows the shared key. If a protocol provides both implicit key authentication and key confirmation, it is called ‘authenticated key agreement with key confirmation protocol’ (AKC) and it performs ‘mutual authentication’.

Security analysis of protocols define two different types of attacks: Adversaries, which perform an off-line analysis of network data and do not interact with any of the communicating parties, conduct a *passive attack* on the protocol. In an *active attack* the more powerful adversary can modify, insert and delete messages transferred over the network. Of course a secure protocol should be able to withstand both types of attacks, as it is reasonable to assume that an attacker has those capabilities in a network.

Over time a certain set of desirable security attributes of a secure protocol have been determined:

- *known key security*. The key exchange should always produce a unique secret key, even if an adversary has knowledge of previously exchanged secret keys.
- *(perfect) forward secrecy*. If a long-term private key is known to an attacker, he will not be able to determine secret keys which have been exchanged by honest parties.
- *key-compromise impersonation*. If host A's long-term private key is known to an adversary C, C can impersonate A now. If key-compromise impersonation is met by the protocol, C can not impersonate other entities to A.
- *unknown key-share*. Host A can not be misled to share a key with entity B without A's knowledge. It is not possible, that A believes it shares the key with C and B believes that the same key is shared with A.
- *key control*. No party should be able to force the session key to some preselected value.

2.3.3 2-party key exchange protocols

The first 2-party key exchange protocols did not provide authentication. They only solved the problem of secure agreement on a shared secret key without having any predistributed information like a password on the communicating hosts.

The first key exchange algorithm based on the discrete logarithmic problem was presented by Diffie and Hellman in 1976 [20]. Their method does not need the generation of a key pair. It communicates a secure key exchange between two peers and provides complete security of the final key from eavesdroppers.

If hosts A and B want to agree on a shared key k , the algorithm works in the following way (presented in the RSA cryptographic system):

1. A and B agree on large primes n and g such that $g < n$
2. A chooses randomly x and sends $X = g^x \pmod n$ to B
3. B chooses randomly y and sends $Y = g^y \pmod n$ to A
4. A calculates $k = Y^x \pmod n$
5. B calculates $k = X^y \pmod n$

A and B agree on the shared key $k = g^{xy} \pmod n$. No adversary listening to the communication is able to determine K , but the protocol does not protect from man-in-the-middle attacks, as no authentication of a party is performed.

The algorithm was also modified to a version based on elliptic curve cryptography by Miller [46] and Koblitz [34]. Another protocol was introduced by Needham

and Schroeder [50] who optionally included an authentication server in their key exchange protocol. The authentication server acts as a trusted third party and owns a public key pair which is known by all communicating peers. Through the authentication server each host can be authenticated and the key can be securely exchanged.

Diffie and Hellman modified their protocol to provide authentication: Instead of choosing random keys, the hosts use the private static key w included in their long-term private/public keypair instead of the values x and y for authentication. Then $X = g^x \pmod n$ would be equal to their public key W . This protocol always results in the same shared key k and does not provide perfect forward secrecy. The same shared key is always used as long as one of the hosts' long-term keypair is not changed, so cryptanalysts have more data to break the key. The protocol is called the static version of the Diffie-Hellman key exchange.

One of the most popular authenticated 2-party key exchange protocols is MQV by Menezes in 1995 [44]. It is based on the Diffie-Hellman key exchange and requires all communicating parties to have a unique public key-pair. The authors assumed, that all hosts know the public key of their communication partner to be able to verify their authenticity. If they do not know it, they will have to verify the authenticity of their partner's certificate which contains the public key. The authors also presented a version with key confirmation called MQVKC. Law et al. revised the protocol in 1998 [37] and extended it for ECC. In 2001, Kaliski [31] discovered that MQV is insecure to an unknown key-share attack and Krawczyk improved the protocol further in 2005 [35]. Leadbitter and Smart showed in 2003 [38] that the ECC version of MQV can be exploited to gain knowledge of one of the hosts' private key. MQVKC has not been shown insecure so far.

Popescu introduced the authenticated key agreement protocols AKAP and a simpler version called SAKAP for ECC in 2005 [54]. He compared the security and performance of various elliptic curve protocols including MTI/A0 and MTI/C0 by Matsumoto et al. from 1986 [42] which has been shown vulnerable by Law [37], one for three parties by Joux from 2000 [28] and one based on the Weil pairing by Smart from 2002 [61]. Strangio also proposed a new ECC authenticated key agreement protocol called ECKE-1 in 2005 [65]. His protocol overview includes additionally the protocols UM from Ankney et al. in 2003 [3], LLK from Lee et al. in 1998 [39], SK from Song and Kim in 2000 [62] and SSEB from Al-Sultan et al. in 2003 [1]. A summary of both overviews can be found in table 3.2.

2.3.4 Multiparty key exchange protocols

Multiparty key exchange adds several new challenges compared to the 2-party key exchange. Mutual authentication requires much more effort in a party with more

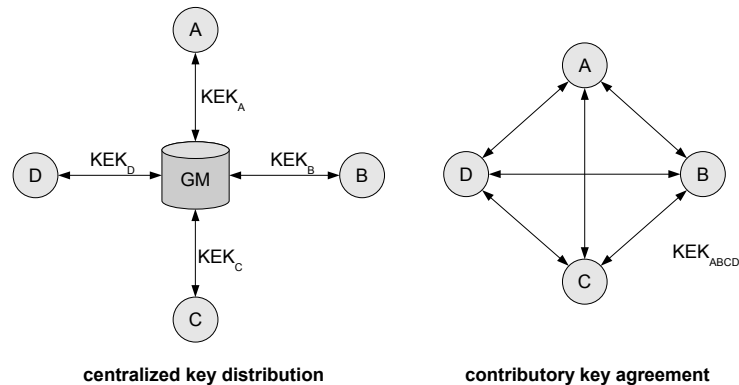


Figure 2.7: different types of multiparty key exchange

than 2 hosts. Dynamic membership, where hosts can join and leave the group dynamically, adds additional complexity to the protocol.

Multiparty key exchange protocols usually support two types of key agreement: In *initial key agreement (IKA)*, the session key is established for the first time in a new group. When the membership changes and a host joins or leaves the group, *auxiliary key agreement (AKA)* is performed. AKA ensures, that former party members do not know the shared key after they have left (forward secrecy) and that old shared keys are not retrievable for new party members (backward secrecy). After every membership change, a new shared key is exchanged.

Centralized key distribution A host inside the group authenticates all members of the group, generates all keys and distributes them to the other members. This central host can either be a trusted third party like a dedicated server, or a chosen group member. It will be called group manager (GM). This host needs to share symmetric keys called KEKs with all group members to distribute the GEK. A 2-party key exchange protocol is needed for that purpose. Figure 2.7 shows the topology. Many protocols are available with this straightforward approach of group key management. It is the preferred choice for real-world applications at the moment. Harney proposed a complete centralized protocol with authentication including IKA and AKA operations called GKMP in 1997 [23]. His approach does not trust a single system’s random number generator, so all keys are generated by the key manager and the group’s first member. Mitra introduced a protocol framework called IOLUS in 1997 [47]. He splits the group into several subgroups which are managed independently, so there is no common global GEK. The protocol relieves a central group manager from communicating with all peers

inside the group and thus makes it more scalable.

Centralized key distribution has several problems caused by its structure: The GM is a single point-of-failure. If the GM is not available any more, the whole group's communication will be interrupted, as new keys can not be distributed any more. The GM could also be a performance bottleneck, especially if no dedicated server is used for the task. Embedded devices often lack the computational resources and network bandwidth to serve several other peers.

Contributory key agreement Contributory key agreement was introduced to solve the problems of centralized key distribution. Its shared session key, the KEK, is derived by all group members where each member contributes the same amount of information and nobody has any advantage over others (see figure 2.7). Thus no member is able to predetermine the resulting value of the KEK on their own. This type of key agreement ensures randomness of the established KEK and the generated random keys are theoretically more secure, as the randomness of all the group members are combined.

One of the first contributory approaches was introduced by Steiner in 1996 [63]. He extended the 2-party Diffie-Hellman key exchange to the n -party case without authentication. His protocols GDH.1 and GDH.2 have two stages: In the upflow stage, user i receives a message, adds his own calculations and resends it to user $i + 1$. When user n receives the final message of the first stage, he also adds his own calculations and sends the result to all other members in the downflow stage. After that each member can compute the shared secret. Those protocols are computationally very expensive, as each member has to perform several asymmetric computations sequentially and the number of rounds is n . Contributory key exchange protocols which are derived from Diffie-Hellman key exchange are called *Group Diffie-Hellman protocols (GDH)*. Several extensions of Steiner's group Diffie-Hellman protocols have been invented over the years, including one from Burmester and Desmedt [12]. It consists of only three rounds of communication, but the authors assume, that the underlying network is capable of simultaneous packet multicasting. If it was not available, all messages would have to be sent to every other member manually and this would result in communication costs linear to the group size.

Kim et al. proposed a tree-based approach called TGDH in 2000 [32]: they established a Diffie-Hellman shared secret by using a binary key tree (see figure 2.8). Every leaf inside the tree represents one party member and every internal node corresponds to two children nodes which perform a Diffie-Hellman key exchange to establish a key for the subgroup below the node. After the subgroup's successful key exchange the resulting key is distributed to the other members of

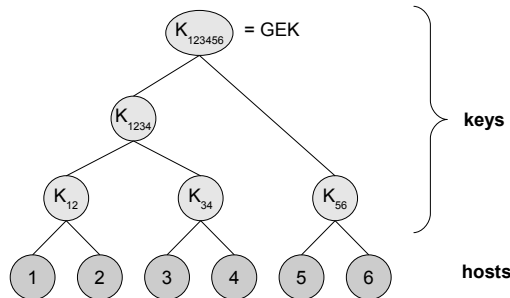


Figure 2.8: structure of TGDH by Kim [32]

the subgroup. This approach reduces the amount of communication rounds can be reduced from n to $\lceil \log n \rceil$. In 2004, Kim et al. improved TGDH and introduced STR (skinny tree) [33]. There the underlying binary tree is completely unbalanced and stretched out. Surveys and comparison of non-authenticated group key exchange protocols including contributory and centralized key exchange are presented by Rafaei and Hutchinson in 2003 [56] and Amir et al. in 2004 [2].

So far none of the presented contributory group key exchange protocols support authentication of the members, although this is one of the most important aspects in secure communication. Early suggested authenticated group key agreement protocols have been shown vulnerable to active attacks [29, 4, 5, 53].

As one of the first, Bresson et al. tried to prove the security of his authenticated group key agreement protocol called AKE1 [8]. The protocol is based on group Diffie-Hellman protocols and requires n rounds to establish the shared key among n users. He produced a mathematical model which defines an environment where an adversary can control all peers and can initiate protocol runs between any peers. The authors issued two following papers which additionally covered dynamic membership changes within the protocol [9, 10].

Yang and Shieh proposed another contributory protocol with authentication in 2001 [70]. It uses a tree-based structure and runs in $\log n$ rounds of communication. The underlying asymmetric cryptographic scheme for mutual authentication is ID-based [60]. There, a trusted third party generates a master public key pair and its public key is known to all hosts. Every party gets an identity string and a corresponding secret key by the third party in an one-time initial setup phase (comparable to key signing by a CA). Using the master public key, the identity string and the secret key, every host can be authenticated.

Joux and Yung exploited pairings of points in elliptic curves which enabled their protocol to run in a single round only in 2000 [28]. This protocol is only

suitable for three parties. Bresson et al. proposed a contributory key agreement protocol with a trusted third party in 2003 [11]. The protocol is designed to run on low-power mobile devices and the third party performs necessary computationally expensive operations. Needed asymmetric operations on mobiles can be precomputed. Nam et al. discovered security flaws in Bresson's protocol in 2005 and proposed an improved version which fixes the flaws [49].

Key predistribution Blom and Matsumoto et al. introduced key predistribution systems in 1984 [6] and 1988 [43]. In these systems a trusted third party distributes secure keys as one time setup. Then any subset of peers can establish a shared secret with efficient symmetric algorithms using those keys. The resulting shared keys are determined a priori by the distributed secure keys. This system offers the advantage, that not all participating nodes have to be online at the same time and thus it is suitable for ad-hoc networks. Ad-hoc networks do not have a fixed infrastructure, they are self-creating on the spot (usually wireless) and routing is performed on the nodes themselves. Key predistribution only provides group authentication in contrast to individual peer/mutual authentication. Chan [14] eliminated the reliance on the trusted third party and introduced a distributed key predistribution scheme.

2.3.5 PKI

All presented protocols which provide authentication rely on asymmetrical key-pairs. This requires a PKI with a trusted third party, called a CA, which guarantees the identity of all hosts. This section gives a short overview to solve the problem of host authentication with as few calculations as possible.

The signature of a host's certificate has to be checked by the hosts which want to validate their peer's identity. Thus at first, certificates have to be exchanged in the protocols, so that every party knows the public key of the peer it communicates with. This step is easy to realize. The actual verification of the certificate by signature verification is a harder problem. This process using ECDSA takes two multiplications. As multiplications are computationally very expensive, this step would significantly slow down the key exchange. There are several options to avoid the additional delay, but we present only a selection:

- All devices cache the certificates which are to be trusted. In a preliminary step, a TTP, e.g. the CA, transfers all trusted certificates to the hosts which store them permanently. When a host needs to be authenticated, its certificate is searched in the internal cache. If a match is found, the host is authenticated. If no match is in the cache, the host is either rejected or the certificate is authenticated by verifying the signature.

- When a host starts communicating with a peer, the peer's public key is retrieved from a TTP. As the public key is sent by the TTP, the host assumes it to be authentic and accepts the peer's identity. This approach always needs access to the TTP to establish a successful secure communication link.

The first option with cache requires a large permanent internal memory and continuous cache updates by the TTP to avoid as many signature verifications as possible. The latter option requires a TTP to be present at all times. Of course also a combination of both approaches would be feasible.

Centralized secure group protocols offer an advantage for authentication purposes when trusted keys are cached on the devices. Each member of the group only has to authenticate the GM and no other member, as they trust the GM. The GM needs to check the identities of all the group's host, so the problem of certificate distribution is limited mainly to the GM. Devices, which will never be a GM only need to cache the certificates of possible GM hosts.

Especially on mobile phones other authentication methods are also conceivable. Every device own a unique ID and is identifiable by its phone number. Usage of different networks, like text messaging, provide computationally cheap means for identification. This could speed up certificate validation significantly, as no asymmetric operations have to be performed then.

The final solution of the authentication/certificate validation problem depends on the architecture of the particular application.

Chapter 3

Secure group key exchange

The goal of this thesis is to find an efficient and secure multiparty protocol for use in mobile devices with limited computational capabilities and without using a trusted third party. At first we will perform an analysis of secure protocols, both for two and more parties. Then we will analyse asymmetric cryptographic algorithms, as they play an essential part in key exchange protocols and authentication. Based on the result of the previous parts, several key exchange protocols are compared in terms of both security and efficiency on mobile devices.

3.1 Analysis of secure protocols

In chapter 2 we presented an overview of available secure protocols including key exchange. To be able to filter the most relevant protocols, we introduce a list of considerations for the choice of an adequate secure protocol.

3.1.1 Considerations

In the ‘Handbook of Applied Cryptography’ from 1996 [45], Menezes et al. explained several design issues for secure protocols. Those issues are related to the application in mobile devices now.

Use of trusted servers A protocol can rely on a trusted third party (TTP). This external party does not actively contribute bulk data to the communication but helps establishing and managing the secure communication. It can either be the group manager in a centralized key distribution scheme or can only be used as authentication server as e.g. in the Needham-Schroeder protocol [50]. Computationally expensive operations could be sourced out to a trusted server and relieve slower mobile devices.

TTPs pose a serious limitation on the usability of the protocols: TTPs always need to be online for operation of the secure group protocol. If it is not available,

no group communication between the devices is possible at all. This work tries to select a protocol which does not rely on a TTP.

Network topology The most used network topology in group communication today is the star. One central server retains secure communication links to all members of the group. If one member wants to send data to the other peers, the data will be sent to the central server, which sends a copy to all the other connected members. This method relieves the group members from the burden of multiple encryptions and multiple message transfers. However, the central server has to be available as long as there are performed secure group communications. This is a severe limitation of the environment and is sometimes not possible to achieve. Mobile devices do not have the capabilities to replace the central trusted server, so computational and network related costs should be distributed evenly between all the members.

We will use the fully meshed net as network topology. Every host sends data to all the other hosts inside the group. No central server is required in this system. Then hosts have to establish secure communication links to all the other peers and encrypt data multiple times. By using a group key agreement protocol, all hosts share a common secret key for data encryption. This reduces the computational cost to a single encryption for the whole group.

Efficiency A protocol's efficiency can be measured in several regards:

- *number of message exchanges.* The amount of messages exchanged between hosts is an important concern, as the transfer latency can be very noticeable in wireless networks.
- *required bandwidth.* Although key sizes of used cryptographic algorithms are rather small (usually below 1 Kbit) certain key exchange protocols require substantial bandwidth caused by their structure, when e.g. no multicasting is available and data has to be sent multiple times to several hosts. Mobile wireless networks only offer limited bandwidth, and so in some cases the data transfer can take significant time.
- *complexity of computations.* Mobile devices offer very limited computational performance. Key exchange protocols rely on cryptographic algorithms which may have substantial computational costs, so the amount of computational costs has to be included in the analysis. Some protocols offer precomputation of complex calculations. This method may be useful for some mobile devices, but operating systems of mobile phones do not offer multithreading. During computation the device will stall and be unusable.

We will not consider any possibilities of precomputation, as we want to find a protocol which minimizes the total amount of calculations. Precomputation will only bring the calculations and the device's stall forward in time. Careful analysis must be conducted, as the development of fast secure protocols is very difficult and unconsidered efficiency improvements in secure systems can significantly lower its security and easily create weaknesses. It is very difficult to ensure security and improve efficiency in the cryptographic area.

The above points have to be well considered in secure group protocols that have to run in mobile devices. This work tries to find a protocol as efficient as possible, but its security must not suffer from this condition.

Resistance against Denial-of-Service attacks Denial-of-Service (DoS) attacks can be performed on hosts which offer the services to clients. There are two types of attacks: Memory-DoS attacks cause the attacked host to use up all available memory. Computational-DoS attacks induce the attacked host to perform a huge amount of calculations to keep it from performing other services. It is desirable that a protocol provides countermeasures to these attacks, but other protection mechanisms like firewalls are also efficient against those attacks. Our considered protocol does not need to have this attribute.

Protocol network layer Network communication can be structured using the OSI layer. IP lies in layer 3 and the upper level protocols TCP and UDP can be found in layer 4. Mobile devices usually provide the TCP/IP and UDP/IP protocol for communication. Therefore the desired protocol should lie, like SSL, in layer 6 or layer 7. IPsec's layer 3 can not be used in mobile devices, as their operating system do not offer the possibility to implement a protocol in layers 1 to 4.

Member authentication Many key exchange protocols offer security without authentication of the participating hosts. In those cases, hosts are not assured that their peers' claimed identity is true. Protocols encrypting data without authenticating the group's members leave a major security flaw exploitable by adversaries. This is why authentication is regarded as a major requirement for secure protocols in this thesis.

There are several methods for authentication available: One of the simplest and fastest methods is password-based authentication. It completely relies on symmetric cryptographic algorithms and does not require expensive computations. Hosts which want to join a communication session have to know a certain password. If it is known, the other peers assume that the host is authorized to join the session.

No mutual authentication of each peer is performed in this scheme. It uses group authentication and trusts every member inside the group that its claimed individual identity is valid. This security level still provides too many points of attack. Thus a protocol should offer mutual authentication.

This will be the case if a trusted third party as authentication server is available. Each host would own a password (comparable to the private key in public-key cryptographic systems) which is also known by the authentication server. The hosts transmits his password to the authentication server and the server confirms the host's identity to the other peers. That scheme requires a trusted third party to be online for authentication all the time.

Key predistribution schemes also provide mutual authentication and only rely on symmetric encryption. A trusted third party initially distributes keys to all hosts who want to authenticate themselves. The TTP is not required for the authentication process itself. Predistribution schemes produce predetermined resulting keys in the key exchange. This fact could make cryptanalysis of the protocol easier as more encrypted data is provided that uses the same key. Additionally, several desirable security attributes are violated by this approach. It is suitable for ad-hoc networks and very slow embedded devices, but not secure enough for the targeted applications in this work.

Asymmetric cryptography using certificates is one of the most secure authentication methods in use today. Every host owns a certificate and a trusted third party, called a certification authority (CA), confirms the authenticity of certificates by signing them in an initial process to set up the public-key infrastructure (PKI). To confirm the validity of a certificate, a host validates the CA's signature of the certificate. If it is valid, the certificate is supposed to be authentic and the host which knows the corresponding private key is authenticated and trusted. Several asymmetric cryptographic algorithms are available for this task, e.g. RSA, ECC or XTR. RSA requires considerable computational effort for keypair generation and decryption. ECC and XTR promise to have lower computational costs, but they are also not guaranteed to perform adequately on low-power mobile devices. As the level of security with public-key cryptographic methods is very high, this is the desired method of authentication.

Group management One peer inside the group must manage it in terms of access control (e.g. who is allowed to enter the group). If a contributory key agreement protocol is employed, every host will be considered equal. There the choice can be arbitrary, but the most reasonable selection would be the host which was the group's first member and which was contacted by the second member. In centralized key distribution protocols the central entity, the GM, should perform

access control, as all the authentication happens there, too. There are several choices available, but in the end this choice depends on the application which performs the group communication and does not depend on the selected key exchange protocol.

Maturity of protocol Secure protocols have a certain evolution: When they are invented and presented, their security can not be proved, as they are usually based on a logical structure which is very difficult (and sometimes impossible) to prove to be secure. After the initial proposal, cryptanalysts search for flaws in the new protocol. When several years passed by and no flaws have been found, it is generally accepted as secure and can be used in practice. Only techniques with a certain age should be used in secure applications. If possible, we only consider protocols with an maturity greater than 5 years.

3.1.2 Assumed conditions

The range of possible applications available is too wide to be able to choose a single best protocol for them. This work is based on some assumed conditions to limit the range of possible network topologies and protocols. We tried to select the conditions in a way, that the analysis still applies to most applications.

- *Network topology: fully meshed.* All hosts can send bulk data to all of the other hosts. A single key should be used for encryption and decryption of bulk data by all hosts. This key is called group encryption key (GEK). That topology is implicitly assumed by most group key exchange protocols, as all hosts of the group receive the same GEK. A topology in a star formation would not benefit from a group key agreement protocol, as only 2-party communication is used there.
- *mature secure protocol.* The protocol is mature, cryptanalysis have worked on it for several years and no security flaw have been found.
- *honest players.* All participating peers always follow the protocol specification and they delete all internal data when terminating. Most of the key exchange protocols either do not end correctly or do not result in a properly exchanged shared secret if one of the peers behaves maliciously and does not follow specifications. Existing protocols like SSL and IPsec make this assumption.
- *secure random numbers.* Random numbers generated on hosts are assumed to be random enough for secure operations. There is no need to combine

randomness from several peers to generate random numbers for a sufficient security level.

- *secure underlying cryptographic principles.* All used cryptographic algorithms like symmetric and asymmetric cryptographic algorithms are secure, e.g. the discrete logarithm problem of RSA, ECC and XTR.
- *dynamic membership.* There is no static group. Members can join and leave the group dynamically at all times.
- *small groups.* Only small groups will be regarded in this thesis. One of the main applications of secure protocols will be conferencing. We assume, that in this use case at maximum 10 members will communicate with each other.

3.2 Asymmetric cryptographic algorithms

We determined in the last section that asymmetric cryptographic algorithms provide very good authentication mechanisms in combination with certificates. However, those algorithms are computationally very expensive and it is not known if they perform fast enough to run on low-power mobile devices like mobile phones. ECC and XTR claim to have a big performance advantage over RSA, so it is likely that they perform quickly enough on the target platform.

In this section the operation of several different asymmetric algorithms are explained and discussed. After that we will describe the implementation of our benchmark application for mobile phones.

3.2.1 Overview

Three different asymmetric cryptographic algorithms will be addressed in this thesis: RSA, ECC and XTR. There are some other algorithms available, but those are not considered to be mature and security concerns prevent them from being used.

The main focus will be put on ECC and XTR, as those algorithms are supposed to perform much faster than RSA. RSA is only included for comparison. Three different operations will be analyzed for each of the protocols:

- generation of a keypair consisting of a public and a private key
- encryption of data
- decryption of data

Only a small amount of data will be encrypted and decrypted in the benchmark. The algorithms only can encrypt and decrypt a number between 0 and a high

value related to the used key size, so all data, that should be encrypted, has to be converted to a value in this range.

RSA

RSA was invented by Rivest, Shamir and Adleman in 1978 [58]. It was the first proposed asymmetric algorithm. Many certificates and authentication methods in use today perform RSA computations. RSA is based on modular integer operations and its underlying infeasible mathematical problem is called the *discrete logarithmic problem*.

RSA makes use of a public parameter N which sets its maximum key size and the security of the system. N defines the modulo value of the system. All calculations are performed modulo that value. If several key pairs are used in one RSA calculation, the key pairs will have to rely on the same N .

The following steps show generation of a keypair for RSA without using a predefined N :

1. Select two large primenumbers $p \neq q$.
2. Calculate $\varphi = (p - 1)(q - 1)$ and $N = pq$
3. Choose an integer e in the range $]1; \varphi]$ which does not share a factor except 1 and -1 to φ
4. Compute d such that $de \equiv 1 \pmod{\varphi}$

Then the public key contains N and e and the private key contains N and d . Generation of the primes p and q has very high computational costs and takes the most time of the keypair generation operation.

Encryption in RSA requires only one exponentiation:

$$c = m^e \pmod{N}$$

The cleartext m , represented as a number in the range $[0; N]$, is risen to the power of the public key exponent e and results in the corresponding ciphertext c .

Decryption is performed similar to encryption:

$$m = c^d \pmod{N}$$

The ciphertext c is risen to the power of the private key exponent d and returns the cleartext m . Most popular implementations choose a small public exponent e and a large private exponent d . Then encryption is much faster than decryption, as encryption uses the small value e for exponentiation and decryption uses the larger value d .

ECC

ECC is based on elliptic curves, which are introduced in chapter 2. Domain parameters define the parameters of an elliptic curve used for the ECC calculations. Those domain parameters will have to be identical, if ECC keys are to be compatible.

Keypair generation is described by the following steps:

1. Select integer w randomly in the range $[1, n - 1]$
2. Calculate $W = wP$

n is the order of the used elliptic curve and P is its basepoint. Both values are defined in the domain parameters. The private key consists of the integer value w and the domain parameters. The public key includes the point W on the elliptic curve and the domain parameters.

There are several methods for encryption with ECC. We will use the Elliptic Curve Integrated Encryption Scheme (ECIES) introduced by Bellare and Rogaway and which has been standardized in ISO/IEC 15946-3 and ANSI X9.63. It is based on the ElGamal public-key encryption scheme. The following process realizes encryption:

1. Select k randomly in the range $[1, n - 1]$
2. Compute $R = kP$ and $Z = hkW$. If $Z = \infty$, go to first step
3. Compute $(k_1, k_2) = KDF(x_Z, R)$, x_Z is the x -coordinate of point Z
4. Calculate $c = ENC_{k_1}(m)$ and $t = MAC_{k_2}(c)$

Parameters n , P and h are provided by the domain parameters and W is the public key of the host which needs to decrypt the data later on. KDF is a key derivation function based on a hashing algorithm which produces two distinct keys. ENC_{k_1} represents a symmetric encryption cipher taking k_1 as key for encryption. MAC_{k_2} is a message authentication code algorithm and uses k_2 as its key. The encrypted plaintext m is located in c , and t is the authentication tag, so that the decrypting host can validate the authenticity and correctness of the calculation. The algorithm sends R , c and t to the decrypting host.

ECIES decryption is described below:

1. validate R as public key
2. Compute $Z = hR$. If $Z = \infty$ reject ciphertext.
3. Calculate $(k_1, k_2) = KDF(x_Z, R)$, x_Z is the x -coordinate of the point Z

4. Compute $t' = MAC_{k_2}(c)$. If $t' \neq t$ then reject ciphertext.
5. Compute $m = DEC_{k_1}(c)$.

The algorithm receives R , c and t from the encrypting host and the domain parameters are already known. KDF and MAC are identical to the functions used for encryption and DEC is the corresponding decryption function to ENC . The algorithm results in the plaintext m if all verifications were successful. Encryption and decryption do not need to use the MAC for authenticity verification. If data is encrypted with a different key, the resulting cleartext will be different after decrypting. The different key on both hosts will prevent further successful communication. We did not test the $MACs$ in the benchmark, as they rely on hashing and thus do not have a relevant performance impact compared to asymmetric cryptographic operations.

Multiplications are the most computationally expensive operations in ECC. Encryption performs two and decryption one multiplication. All other operations in the ECIES encryption and decryption algorithm require only marginal computational time.

XTR

XTR is a more efficient version of RSA using traces. Like RSA, it is also based on the discrete logarithm problem.

The following steps produce a keypair:

1. Select w randomly in the range $[1; q - 2]$
2. Compute $W = g^w \pmod p$

q is the subgroup order of the trace, p is the modulo value and g is the predefined basepoint for calculations. All of the three values have to be predefined and used by all certificates which have to be compatible (similar to the domain parameters for ECC). The above algorithm results in the keypair with private key w and public key W .

Lenstra and Verheul described in their paper [41] an XTR-ElGamal encryption algorithm. It is similar to the ElGamal encryption algorithm and thus also similar to ECIES. Encryption works as follows:

1. Select k randomly in the range $[1, q - 2]$
2. Compute $r = g^k \pmod p$
3. Compute $z = W^k \pmod p = g^{kw} \pmod p$

symmetric algorithms	RSA	ECC	XTR
80	1024	163	170
128	3072	283	512
192	7680	409	1280
256	15,360	571	2560

Table 3.1: key sizes of similar security levels for different cryptographic algorithms

4. Compute $k = KDF(z)$
5. Compute $c = ENC_k(m)$

The parameters q , p and g are public. W is the public key of the host which needs to decrypt the resulting data. KDF is a key derivation function based on a hash algorithm which generates a key for symmetric encryption in the desired size. ENC is a symmetric encryption algorithm which encrypts the plaintext m . Required values for decryption are r and c .

Decryption of the ciphertext c is performed in the following way:

1. Compute $z = r^w \pmod p (= g^{kb} \pmod p)$
2. Compute $k = KDF(z)$
3. Compute $m = DEC_k(c)$

r and c are the output of the encryption algorithm. KDF is the same function as before and DEC is the corresponding symmetric decryption algorithm to ENC . m contains the plaintext.

Like ECC, XTR-ElGamal encryption needs 2 exponentiations for encryption and 1 exponentiation for decryption. Thus, encryption needs roughly twice as long as decryption.

Key size comparison

RSA, ECC and XTR use different underlying mathematical concepts to provide security. This fact results in different key sizes for a similar security level. A security level is defined as a measure, how much difficulty it causes to break a cipher with a certain key size.

Table 3.1 compares the key sizes of symmetric, RSA, ECC and XTR algorithms for different security levels. It is based on data from Lenstra [41] and from Lauter [36]. RSA based encryption are considered secure with 1024 bit key sizes at the moment. They are as secure as an EC cryptographic algorithm with a much lower key size of 163 bit and XTR with a key size of 170 bit. Symmetric ciphers are in

comparison more secure, they only need 80 bit keys. The higher the key size of an algorithm, the more calculations are required in its operations. That explains why XTR and ECC gain performance advantages through the use of lower key sizes. As mobile devices have very limited computational capabilities, we will not test higher key sizes than the equivalent of 1024 bit for RSA.

3.2.2 Benchmark

We implemented a benchmark which tests the performance of the asymmetric cryptographic algorithms RSA, ECC and XTR. We tested the operations key generation, encryption and decryption with different key sizes.

The benchmark is implemented in Java as a MIDlet so that it can run in various mobile phones with MIDP support (see section 2.1.2). RSA and ECC were already implemented in the Bouncy Castle APIs [7]. Those are libraries for Java providing various cryptographic functions. Parts of their implementation were supposed to run on desktop computers, so some code had to be adapted to run them on mobile devices with limited computational capabilities and memory. The ECC implementation is based on ECP. RSA encryption and decryption operations were taken from the Bouncy Castle APIs. Although the EC cryptographical system was included in the APIs, encryption and decryption operations were not supported, so we implemented the ECIES algorithm in the benchmark. The XTR code was taken from the Crypto++ cryptographic API [17]. The code was ported from C++ to Java and the encryption and decryption algorithms were implemented according to Lenstra's original proposal for XTR-ElGamal encryption.

3.3 Comparison of key exchange protocols

We defined important attributes in section 3.1 and identified asymmetric cryptographic algorithms as an ideal method to perform authentication in secure communication. Now we will perform an analysis and comparison of several key agreement protocols which are based on asymmetric cryptography and provide authentication.

All of the presented key exchange protocols agree on a key encryption key (KEK) for several parties. This KEK is not changed as long as no member joins or leaves the protocol. It helps agreeing on short-lived session keys, which are called group encryption keys (GEK) in multiparty communication. The GEK encrypts the application's bulk data and is known by all members of the communication group. It is reasonable to perform as much work as possible by faster symmetric algorithms, as asymmetric cryptographic algorithms are computationally expensive. So the agreement on the KEKs is performed by slow asymmetric

algorithms and afterwards the repeated agreement on the GEK is performed by fast symmetric algorithms. So rekeying operations on the GEK do not require substantial computational resources and can be performed more quickly and more regular.

In the following part the performance of several group key agreement schemes are theoretically analyzed. We want to determine the total time needed to perform certain operations for the protocols. Thereby we assume, that all hosts are identical and perform all calculations with the same speed. Algorithms based on symmetric cryptographic techniques are assumed to take no time for computation, as only very little data (usually encryption keys) is encrypted. The number of asymmetric operations (also called exponentiations) is counted for each operation of the protocol. Previous papers only consider the total number of exponentiations on each host and for each protocol round (as far as the author knows). Those values only give a rough estimation on the time needed for the protocol's execution. In this work, all calculations performed simultaneously on different machines are only counted as one calculation, as in that case no additional time is needed for the whole protocol execution. Time needed for data transfer in wireless networks can also impact the performance of a key exchange protocol. Thus the amount of message exchanges will also be calculated. We do not consider the transmission delay caused by the transfer of larger amounts of data, as usually only small keys and status messages are transferred. So parallel message transfers and messages sent to multiple destinations are counted as a single message transfer. Message transfers which do not have any impact on the time needed for protocol execution are not counted. We use the term no_e for the number of needed sequential exponentiations and no_c for the total number of sequential message transfers which delay protocol execution.

We want to determine the total time needed to perform the mentioned operations for each protocol. We assume, that all hosts are identical and perform all calculations equally fast. Message transfers are also assumed to take a constant time.

Each protocol provides the following operations:

- *initial setup*. This operation is executed when a secure communication session is to be set up. Hosts U_1 to U_n agree on a common KEK at the same time.
- *member join*. A new host U_{n+1} joins a communication group. A member join operation can only be performed, when the group already set up the secure communication protocol. All protocols considered here offer backward secrecy, so the GEK will be changed after the new member has been

authenticated and before the new party receives it.

- *member leave*. A host leaves the group. As the protocols provide forward secrecy, the GEK will be changed after the host has left the group.

If only static membership had been requested, the operation ‘initial setup’ would have been sufficient. We only regard protocols supporting dynamic membership. More complex cases like ‘group merge’ or ‘group split’, where several members join or leave the group at the same time, are not regarded, as the type of target applications does not require them. They can be easily realized by ‘member join’ and ‘member leave’ operations.

All key exchange protocols are analyzed using ECC as underlying cryptographic system.

3.3.1 Contributory key agreement

It is very difficult to construct a secure contributory key agreement protocol for groups. Many earlier suggestions such as the A-GDH.2 protocol suite from Ate-niese [5] have been proven to be insecure.

The original protocols only agree on a KEK. We included GEK generation and distribution by a randomly chosen member to reflect a more practical operation of the protocol. This section presents two contributory protocols: AKE1 and TGDH. AKE1 uses a ring-based structure and TGDH is based on a tree-like structure.

AKE1

AKE1 was introduced by Bresson in 2001 [8] and was further enhanced in his papers from 2001 [9] and 2003 [11]. It is one of the few contributory group key agreement protocols which are still considered secure. The protocol is based on the group Diffie-Hellman key exchange. It is less than 5 years of age, but because of lack of alternatives we still included this protocol in the analysis.

The hosts are arranged in a ring. All exchanged messages are signed using the hosts’ long-term certificate and contain identifying strings. The signatures are verified by the receiving hosts. It is assumed, that the protocol makes use of ECDSA. ECDSA performs one EC multiplication for the signing operation and two multiplications to verify a signature.

Bresson proved, that AKE1 offers the security attribute perfect forward secrecy. He did not elaborate on other attributes.

Initial setup The initial setup is performed in two stages: up-flow and down-flow. In the first stage, x_1 is chosen randomly in a range depending on the keysize

of the underlying asymmetric cryptographic algorithm. Then host U_1 generates the values Z and Y .

$$\begin{aligned} Z &= g^{x_1} \\ Y &= g \end{aligned}$$

This step requires one exponentiation plus one exponentiation for the signing operation of the message. U_1 sends Z and Y to the next host. Every host U_i receives the values

$$\begin{aligned} Z &= g^{\prod_{t \in [1; i]} x_t} \\ Y &= \bigcup_{m \in [1; i]} Z^{\frac{1}{x_m}} \end{aligned}$$

Host U_i verifies the received message's signature and then calculates the values

$$\begin{aligned} Z' &= Z^{x_i} = g^{\prod_{t \in [1; i]} x_t} \\ Y' &= \bigcup_{m \in [1; i]} Z'^{\frac{1}{x_m}} \end{aligned}$$

which require i exponentiations plus 3 exponentiations for signature verification and generation. Values Z' and Y' are sent to U_{i+1} . The last host U_n inside the ring performs the same calculations as the hosts before. The down-flow stage begins when U_n sends Y' to all hosts $U_i, i \in [1; n]$. U_n has to conduct $n - 1$ exponentiations plus 3 exponentiations for signature verification and generation. Every host i receives the value

$$Y_i = g^{\prod_{t \in [1; n] \setminus i} x_t}$$

and receives part of the shared secret key $sk = g^{x_1 \dots x_n}$ (KEK) by calculating $sk = Y_i^{x_i}$. This requires one more exponentiation plus two exponentiations for signature verification on each of the hosts.

All exponential asymmetrical operations in the up-flow stage of AKE1 have to be performed sequentially. The last three exponentiations in the down-flow stage can be conducted in parallel. The total number of sequential exponentiations is shown below. Exponentiations caused by signature generation and verification are marked with a star.

$$\begin{aligned} no_e &= \underbrace{1 + 1^* + (2 + 3^*) + (3 + 3^*) + \dots + ((n - 1) + 3^*)}_{\text{up-flow hosts } 1 \dots n - 1} + \underbrace{n - 1 + 3^*}_{\text{up-flow host } n} \\ &+ \underbrace{1 + 2^*}_{\text{down-flow hosts } 1 \dots n} = \frac{n(n - 1)}{2} + 4n \end{aligned}$$

Every host sends one message to another host during the initial setup phase. The GEK is generated on a host and transferred to the other members in the end, which requires one additional message transfer. Thus the total amount of messages sent are:

$$no_c = n + 1$$

Member join When a new host U_{n+1} joins, host U_n selects a new random value x_n and calculates by using the cached values of the previous protocol operation

$$\begin{aligned} Y' &= \bigcup_{m \in [1;n]} Y_{x_n}^{x'_m} \quad \text{and} \\ Z' &= sk_{x_n}^{x'_n}, \quad \text{so that} \\ Y' &= \bigcup_{m \in [1;n]} Z'^{\frac{1}{x_m}} \end{aligned}$$

and sends Z' and Y' to U_{n+1} . This step requires n exponential operations plus one for signature generation. Host U_{n+1} performs the same operations as U_n in the last part of the initial setup's up-flow stage. After that the down-flow stage and GEK distribution follows, which is identical to the initial setup. Thus the total number of sequential exponentiations no_e and the total number of messages sent no_c are

$$\begin{aligned} no_e &= \underbrace{n + 1^*}_{\text{recalculations of } U_n} + \underbrace{n + 3^*}_{\text{up-flow stage } U_{n+1}} + \underbrace{1 + 2^*}_{\text{down-flow stage with } n + 1 \text{ hosts}} \\ &= 2n + 7 \\ no_c &= 3 \end{aligned}$$

Member leave U_n leaves the group. The host U_i with the highest remaining index i , in the current case $n - 1$, chooses a new random value x'_{n-1} and calculates

$$Y' = \bigcup_{m \in [1;n-2]} Y_{x_{n-1}}^{x'_m}$$

using the saved values of the up-flow stage of the last initial setup. Then the down-flow stage of the initial setup and distribution for the GEK is repeated for $n - 1$ hosts. The total number of sequential exponentiations no_e and the total number of sequentially transferred messages no_c are

$$\begin{aligned} no_e &= \underbrace{n - 2 + 1^*}_{\text{recalculations of } U_{n-1}} + \underbrace{1 + 2^*}_{sk \text{ calculations of } n - 1 \text{ hosts}} \\ &= n + 2 \\ no_c &= 2 \end{aligned}$$

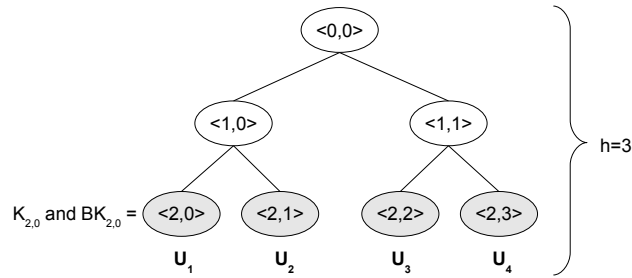


Figure 3.1: sample key structure of TGDH

TGDH

TGDH is a contributory key exchange protocol which is designed to be scalable for larger groups. It was proposed by Kim et al. in 2000 [32]. Its primary structure is based on a fully balanced binary tree. The original protocol does not provide authentication. It is included in the analysis because of its special structure which scales much better than the other contributory protocol AKE1. As only protocols with authentication are considered in this thesis, authentication will be artificially added to the protocol: Every message exchanged between hosts will be signed by the sender and the signature will be verified by the receiving host. The signature keypairs are chosen beforehand and their public parts are known by all hosts. This is only a theoretical measure to be able to compare the protocol with other authenticating schemes. It is not assumed, that the protocol is secure against malicious attacks.

Figure 3.1 shows an example of TGDH's key tree when 4 hosts negotiate one common key. Each member owns a secret key K and a blinded/public key $BK = g^K$. The members are located in the sample tree in nodes $\langle 2,0 \rangle$ to $\langle 2,3 \rangle$. The blinded keys are transferred to all hosts which need them for calculations. All white nodes in the tree represent a keypair which is negotiated between the hosts and the secret key of the root node $\langle 0,0 \rangle$ is the common key which is known by all hosts. It is the result of the negotiation. Every host knows all keypairs of the nodes following the path to the root, e.g. host U_2 knows its own keypair $\langle 2,1 \rangle$ and the negotiated keypairs $\langle 1,0 \rangle$ and $\langle 0,0 \rangle$. This tree structure enables computationally efficient key agreement in larger groups.

Initial setup The KEK is computed from the bottom up: At first, the hosts calculate their pair-wise shared keys in the second level from the bottom (level 1 in figure 3.1). Then the keys of the level above are computed by 4 hosts, and so on, until the root-level with the KEK $\langle 0,0 \rangle$ is reached.

step	host 1	host 2	host 3	host 4
1	choose $K_{2,0}$ $BK_{2,0} = \alpha^{K_{2,0}}$	choose $K_{2,1}$ $BK_{2,1} = \alpha^{K_{2,1}}$	choose $K_{2,2}$ $BK_{2,2} = \alpha^{K_{2,2}}$	choose $K_{2,3}$ $BK_{2,3} = \alpha^{K_{2,3}}$
2		$\xrightarrow{BK_{2,0}}$ $\xleftarrow{BK_{2,1}}$		$\xrightarrow{BK_{2,2}}$ $\xleftarrow{BK_{2,3}}$
3	$K_{1,0} = BK_{2,1}^{K_{2,0}}$ $BK_{1,0} = \alpha^{K_{1,0}}$	$K_{1,0} = BK_{2,0}^{K_{2,1}}$ $BK_{1,0} = \alpha^{K_{1,0}}$	$K_{1,1} = BK_{2,3}^{K_{2,2}}$ $BK_{1,1} = \alpha^{K_{1,1}}$	$K_{1,1} = BK_{2,2}^{K_{2,3}}$ $BK_{1,1} = \alpha^{K_{1,1}}$
4		$\xleftarrow{BK_{1,1}}$	$\xrightarrow{BK_{1,0}}$ $\xrightarrow{BK_{1,0}}$	$\xleftarrow{BK_{1,1}}$
5	$K_{0,0} = BK_{1,1}^{K_{1,0}}$	$K_{0,0} = BK_{1,1}^{K_{1,0}}$	$K_{0,0} = BK_{1,0}^{K_{1,1}}$	$K_{0,0} = BK_{1,0}^{K_{1,1}}$

Figure 3.2: Initial setup of the TGDH protocol for 4 users

Figure 3.2 shows an example for 4 hosts computing the common GEK. It does not include the artificially added signature generation and verification. The following steps are performed:

1. All hosts U_i choose their secret keys $K_{2,i}$ and calculate the blinded keys $BK_{2,i}$.
2. Hosts U_1 and U_2 and hosts U_3 and U_4 exchange their blinded keys.
3. Level 1 keys $K_{1,0}$ and $K_{1,1}$ are calculated by the hosts using the keys which were received from their partner host in the last step. The blinded keys $BK_{1,0}$ and $BK_{1,1}$ are also calculated.
4. Host U_2 sends the blinded key to all hosts of the group of key $K_{1,1}$ as a random representative of the group for key $K_{1,0}$. Host U_3 performs the same step with key $BK_{1,1}$ and transfers it to hosts U_1 and U_2 .
5. All hosts calculate the secret key $K_{0,0}$, which is the resulting KEK.

Like before, the performance estimation does not count parallel computations, only sequential ones. Parallel computations on different devices do not take additional time in the protocol flow. The signature algorithm used for the calculations is ECDSA. Signature generation needs one asymmetrical operation and verification takes two operations.

Calculations in the lowest level of the tree require one asymmetric operation (see step 1 in figure 3.2) and the signature generation also needs one operation. The highest level of the tree requires one exponentiation (see step 5) and two signature related exponentiations for verification. All levels in between require

two exponentiations for the key calculations (see step 3) and three exponentiations for signature verification and generation. Thus the total number of sequential exponentiations is

$$no_e = \underbrace{1 + 1^*}_{\text{bottom level}} + \underbrace{([\log_2 n] - 2)(2 + 3^*)}_{\text{intermediate levels}} + \underbrace{1 + 2^*}_{\text{top level}} = 5[\log_2 n] - 5$$

Numbers marked with a star represent the signature related operations. $\lceil \log_2 n \rceil$ is the height of the tree.

For each layer except the lowest one, one message has to be transferred (not counting several simultaneous message transfers). The final distribution of the GEK using symmetrical encryption uses one more message transfer. Thus the number of transferred messages is

$$no_c = \lceil \log_2 n \rceil$$

Member join When a new member joins the group, it is inserted at a place which does not increase the overall tree height, if possible. The tree needs to stay balanced to obtain optimum performance. All keys from the root to the new member need to be recalculated. This results in the same amount of sequential exponentiations as for the initial setup of the protocol, although the total number of parallel exponentiations is less, as some keys need not to be recalculated. The number of sequential message transfers is also identical to the initial setup. Thus, no_e and no_c result in the following equation for $n + 1$ hosts:

$$\begin{aligned} no_e &= 5\lceil \log_2(n + 1) \rceil - 5 \\ no_c &= \lceil \log_2(n + 1) \rceil - 1 \end{aligned}$$

Member leave The same reasoning like for member joins applies to member leave operations, too. After a member left the tree, its nearest neighbor generates a new secret key and initiates a recalculation of all key nodes up to the root. The number of sequential exponentiations and message transfers is then

$$\begin{aligned} no_e &= 5\lceil \log_2(n - 1) \rceil - 5 \\ no_c &= \lceil \log_2(n - 1) \rceil - 1 \end{aligned}$$

3.3.2 Centralized key distribution

Centralized key distribution relies on one host, which communicates with all other group members, in contrast to contributory key agreement. The central host is called group manager (GM). It can either be one of the group members or a trusted third party. It is the preferred method to perform group key distribution at the

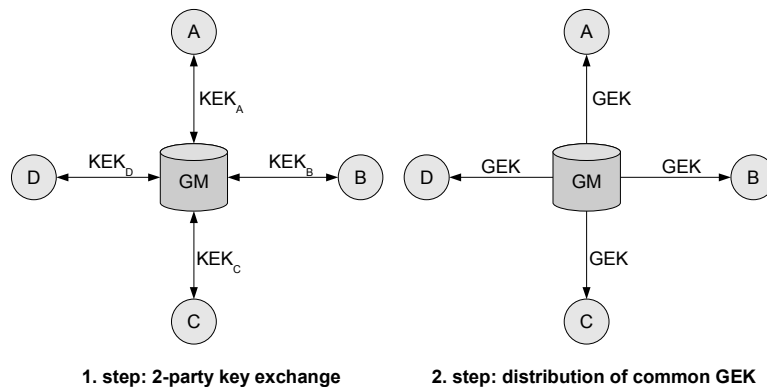


Figure 3.3: Process of key distribution with a group manager

moment. The group members do not perform any key agreement with each other, but only with the central GM. Figure 3.3 shows the process of centralized group key distribution. It involves two steps:

1. The GM performs 2-party authenticated key exchange with all members in the group. As a result, the GM and each of the members share a common key respectively. Those keys are called key encryption keys (KEK), as all further key distribution is performed using symmetric encryption with the KEKs.
2. The GM chooses randomly a common GEK and distributes it to all other members.

The most computationally expensive operations lie in the first step: we require adequate authentication for our protocol, so the 2-party key exchange has to be performed using asymmetric cryptography.

Comparison of 2-party key exchange protocols

The key exchange protocol used in the initial exchange of the KEKs between GM and the other hosts needs to be as efficient as possible. We will present an overview of various authenticated 2-party key exchange protocols.

Strangio and Popescu [65, 54] performed an analysis of current key 2-party authenticated key exchange protocols that use ECC. Their results are shown in table 3.2. It shows the year when each protocol was proposed by its author and the computational complexity regarding asymmetric operations. Values in brackets show necessary online computations when precomputation has already been performed. The table also shows all possible attacks for each protocol which have

<i>protocol</i>	<i>year</i>	<i>complexity</i>	<i>attacks</i>
AKAP	2005	3	none
ECDH	1986	2	MITM
ECKE-1	2005	3 (2)	none
Joux	2000	2	UKS
LLK	1998	2 (1)	none
MQVKC	1995	2.5	none
MQV	1995	2.5 (1.5)	UKS, IKC
MTI/A0	1986	3 (2)	UKS, IKC
MTI/C0	1986	2	SSA
SAKAP	2005	1	none
SK	2000	3 (2)	IKC
Smart	2002	4	FSA
SSEB	2003	3 (2)	IKC
UM	2003	3 (2)	KCI

Abbreviations:

MITM: man-in-the-middle-attack
UKS: unknown key-share attack
IKC: imperfect key control
SSA: small subgroup attack
FSA: forward secrecy attack
KCI: key-compromise impersonation

Table 3.2: Comparison of various ECC 2-party key exchange protocols

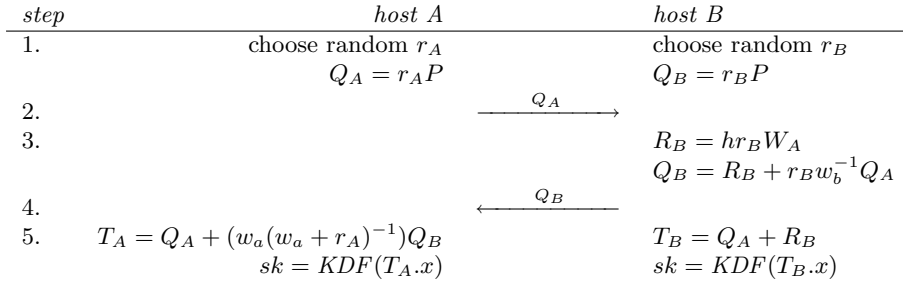


Figure 3.4: The LLK protocol

been discovered. A small subgroup attack enables the cryptanalyst to reduce the search space for one of the private keys significantly if he has partial information about the key. ECDH is the Diffie-Hellman protocol for ECC without any authentication, so it is vulnerable to the man-in-the-middle attack. Aside from Popescu’s and Strangio’s newly proposed protocols AKAP, SAKAP and ECKE-1 only MQVKC and LLK has no known attacks. Almost all of the protocols require 2-3 asymmetrical operations per run. Popescu’s and Strangio’s new protocols will not be covered in the analysis as their age is less than a year.

LLK requires 2 operations and using precomputation, only 1 operation for the key exchange per host. It seems to be secure, as it was proposed in 1998 and no attacks are known yet. The protocol looks promising, so we will present a short overview of it in the next section.

LLK

Overview The LLK key agreement protocol was introduced by Lee et al. in 1998 [39]. The security attributes are supposed to be known-key security, forward secrecy, key-compromise impersonation resilience and unknown key-share resilience. An introduction to the security attributes can be found in section 2.3.2. The protocol provides implicit authentication. The version which is presented here relies on ECC and requires common domain parameters of a negotiated curve which are used on all hosts.

The protocol requires the hosts to own a certificate with public key W and corresponding private key w for authentication. The certificate is known to the peer, too. If it is not known, it can be transferred in an additional step. We will not consider this step in the analysis. h is the cofactor and P is the basepoint of the chosen domain parameters. The following steps are executed (see also figure 3.4):

1. Both hosts choose a random number r_A and r_B and calculate Q_A and Q_B . This step could be precomputed, as it does not rely on a specific peer. It

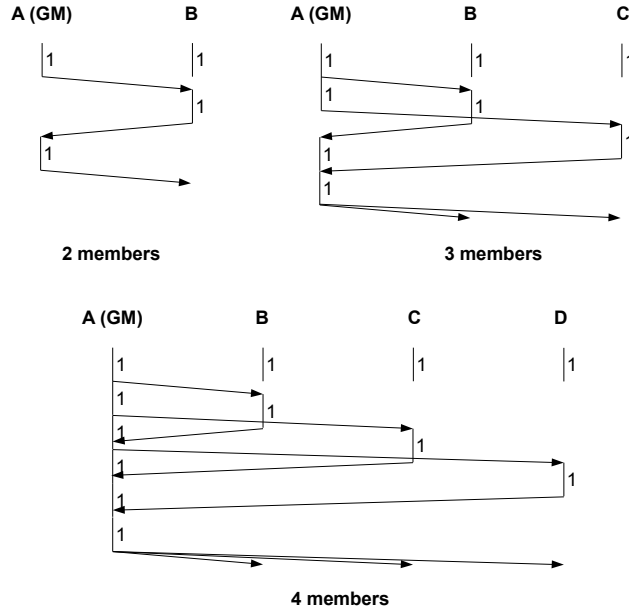


Figure 3.5: Analysis of LLK in an initial multiparty key exchange

requires one sequential multiplication in the elliptic curve.

2. Host A sends Q_A to host B.
3. Host B calculates Q_B which needs one asymmetric operation.
4. Host B sends Q_B to host A.
5. Both hosts calculate the final shared key. Calculation of T_A and T_B is performed using one ECC multiplication on host A. KDF is a key derivation function based on a hash function which returns a shared key sk with an appropriate key size.

T_A and T_B are derived by the formula

$$T_A = T_B = (r_A w_B + r_B w_A)P$$

The protocol results in the shared KEK sk which can be used in further communication using symmetrical encryption.

Now LLK will be analyzed in the multiparty setting. The first host in the group is supposed to be the group manager and performs the key exchange with all other members. New members contact the GM to start the member join operation.

Initial setup Figure 3.5 shows an example of the key exchange for 2, 3, and 4 members in a group. Each vertical line marked with a ‘1’ represents one multiplication in the elliptic curve. Arrows represent the transfer of a message.

In the 2 member setting, both hosts perform initial calculations at first. Then host A sends a message to host B. B conducts calculations on the message’s values. After that the results are sent back to host A, which also performs one multiplication in the elliptic curve. Host A sends the GEK to host B using symmetric encryption in the last step. In total, $no_e = 3$ multiplications are computed sequentially and $no_c = 3$ rounds of messages are sent.

The key exchange with 3 members enables more parallel calculations and reduces the idle time of the group manager. It is obvious, that the GM is the most busy host in the group, as it has to perform most calculations. The total number of sequential multiplications is $no_e = 4$ and the number of message transfers is $no_c = 3$.

The GM has no idle time any more in groups consisting of $n \geq 4$ members. This is based on the assumption, that 2 message transfers take less time than one multiplication in the elliptic curve on the GM. In this case, the total number of sequential multiplications is $no_e = 2(n - 1) = 2n - 2$ and only the last round of distributing the GEK is affected by message transfers, so $no_c = 1$. As a generalization, the idle time of the GM t_i is determined by the following equation:

$$t_i = 2t_c + (3 - n)t_e, \quad \text{if } t_i < 0, \text{ then } t_i := 0$$

t_c is the time needed for transferring one message from one host to the other, t_e is the calculation time of one exponentiation and n is the number of members in the group (including GM). It is not possible to calculate the exact number of asymmetric operations and message transfers without knowing t_e and t_c . So we calculate the execution time including those values. If t_i is negative, GM has no idle time. The total time of initial setup is then

$$t = t_i + t_e no_e + t_c no_c = t_i + t_e(2n - 2) + t_c$$

no_c only includes GEK distribution, so $no_c = 1$ at all times. t_i includes additional delays caused by message transfers.

Member join New members are usually included in the group’s encryptions scheme by performing one 2-party key exchange and sending a new GEK to all members. Thus the total number of sequential multiplications and messages is

$$\begin{aligned} no_e &= 3 \\ no_c &= 3 \end{aligned}$$

Member leave When a member leaves the group, the GEK has to be updated. This is done with a single message from the GM to all other group members. No multiplications in the elliptic curve have to be performed:

$$\begin{aligned}no_e &= 0 \\no_c &= 1\end{aligned}$$

Extension to slower 2-party key exchange protocols Many 2-party key exchange protocols like MTI/A0 [42] or UM [3] need 3 asymmetrical operations per host. We modified the formulas used to calculate the performance with LLK, so that after the first received message on either host two instead of one asymmetrical operations are needed (compare to step 3 and 5 in figure 3.4). Then the number of asymmetrical operations no_e and idle time t_i changes to

$$\begin{aligned}no_e &= 3(n - 1) = 3n - 3 \\t_i &= 2t_c + 2(3 - n)t_e\end{aligned}$$

for initial setup. Member join operations result in the number of exponentiations no_e and number of message transfers no_c

$$\begin{aligned}no_e &= 4 \\no_c &= 3\end{aligned}$$

Member leave operations are identical to the performance of LLK.

3.3.3 Benchmark

We implemented a key exchange benchmark for mobile phones. It makes use of centralized key distribution using the LLK key exchange protocol. Asymmetric calculations are based on ECC with a keysize of 160bit (roughly equivalent to 1024 bit for RSA). The benchmark tests the initial setup operation and simulates member joins and leaves. The key exchange process was measured on one series 60 mobile phone as GM and two series 40 mobile phones as group members. All devices were connected to a GPRS network of the same provider. We used an implementation from the Bouncy Castle API for AES to encrypt and decrypt the GEK for distribution. The key size for AES was 256 bits.

Chapter 4

Results and Discussion

Secure group communication protocols and asymmetric cryptographic algorithms for authentication were analyzed in the last chapter. Now the results of the analysis will be discussed and the benchmarks will be interpreted and compared with each other. As asymmetric cryptographic algorithms play an essential part in secure protocols with authentication, they will be discussed in the first section. After that the key exchange protocols are compared with each other based on the results of the benchmark for asymmetric cryptographic algorithms.

4.1 Asymmetric cryptographic algorithms

The asymmetric cryptographic algorithms RSA, ECC and XTR were benchmarked on mobile phones to determine their feasibility in this limited environment. A series 40 Nokia 6320 and a series 60 Nokia 6630 mobile phone were used for this test. One MIDlet containing the benchmarking program was created and executed on both devices, so that the tests were performed identically. To obtain real world figures, the phones were connected to a phone network, but no other application was being executed at the same time.

Different key sizes were tested for each asymmetric algorithm to get an overview how the performance changes in relation to the key size. The following sizes were used for each algorithm:

- RSA: 1024 bit, 768 bit, 512 bit
- ECC: 192 bit, 160 bit, 128 bit, 112 bit
- XTR: 204 bit, 170 bit, 136 bit, 120 bit

RSA with key size 1024 bit, ECC with 160 bit and XTR with 170 bit roughly provide equal security levels which are considered secure at the moment. Each test was executed 6 times. The average values for key generation, encryption and decryption are shown in figures 4.1, 4.2 and 4.3 for the series 40 mobile phone and in

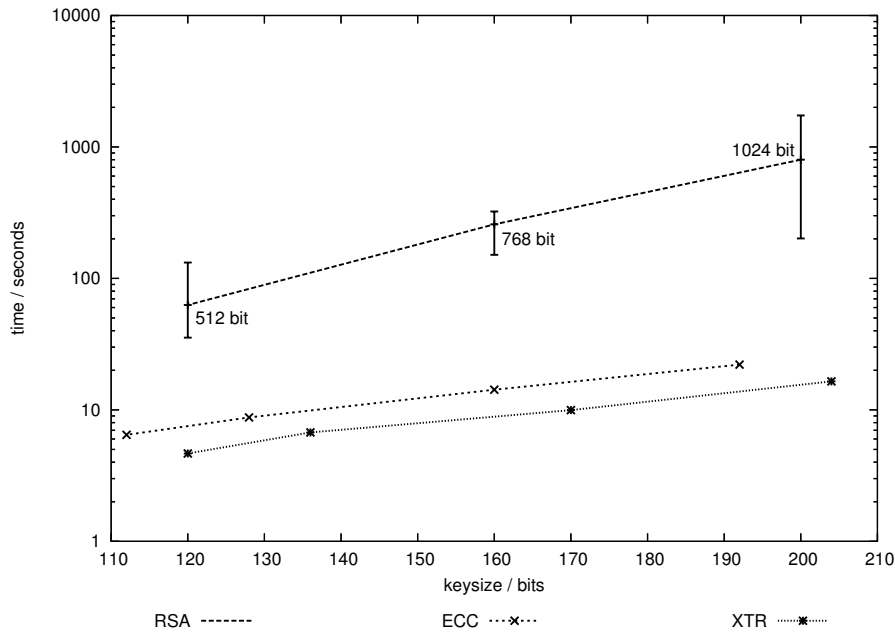


Figure 4.1: Key generation benchmark results in series 40 mobile phones¹

figures 4.4, 4.5 and 4.6 for the series 60 mobile phone. Key generation, encryption and decryption time is displayed for each of the cryptographic algorithms.

RSA uses much higher keysizes than ECC and XTR, so its graph does not relate to the x-axis in the figures. The values for the tested keysizes are shown near the measured times of RSA. Calculation times in the RSA system sometimes showed large variances. Errorbars indicate the maximum and minimum values of the testset.

The test results are summarized in the following sections. We will only discuss the keysizes 1024 bit for RSA, 160 bit for ECC and 170 bit for XTR, as smaller keysizes are considered insecure nowadays and larger keysizes require more computational overhead, which is not readily available on our target platform.

4.1.1 RSA

RSA key generation with a keysize of 1024 bit takes between 3.3 and 28.3 minutes on the series 40 mobile phone and 38.9 to 166.1 seconds on the series 60 device. The large variance of this operation is caused by the trial-and-error mechanism that tries to find an appropriate RSA key pair by random selection of numbers and test for validity afterwards.

Encryption with 1024 bit keys takes 0.135 seconds in a series 40 device and

¹ECC and XTR graphs relate to x-axis, RSA does not relate to it. The corresponding key size is specified at each point of the RSA curve.

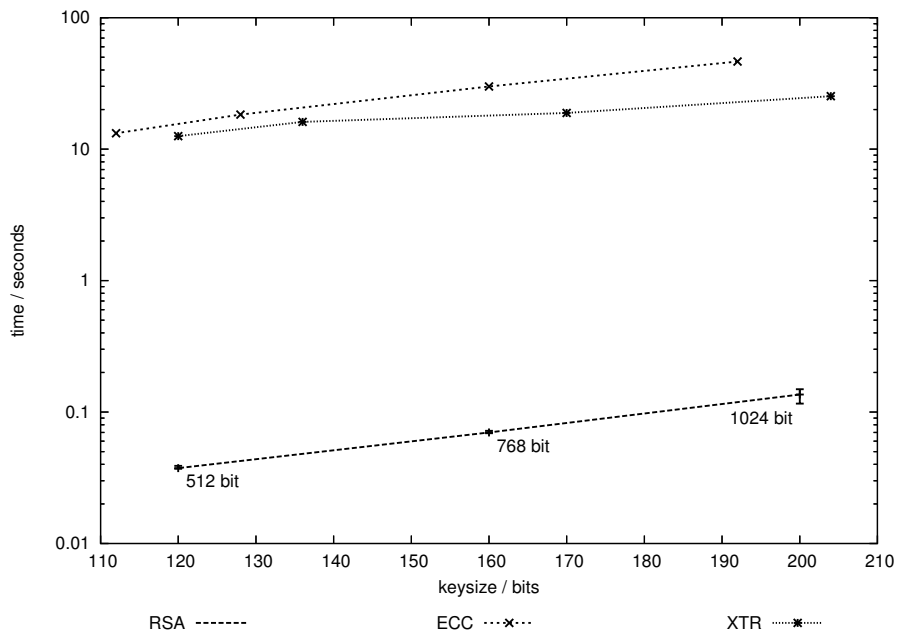


Figure 4.2: Encryption benchmark results in series 40 mobile phones

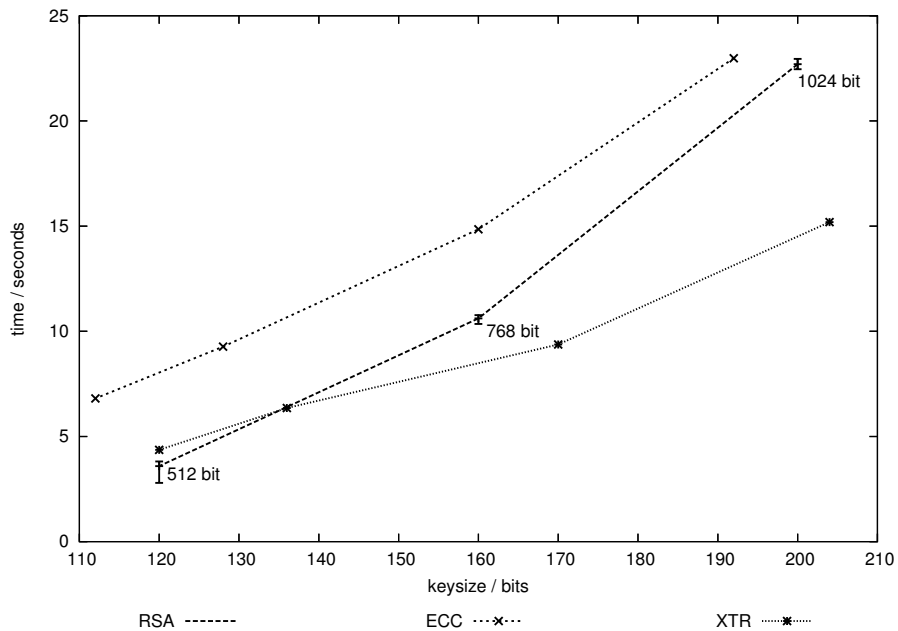


Figure 4.3: Decryption benchmark results in series 40 mobile phones

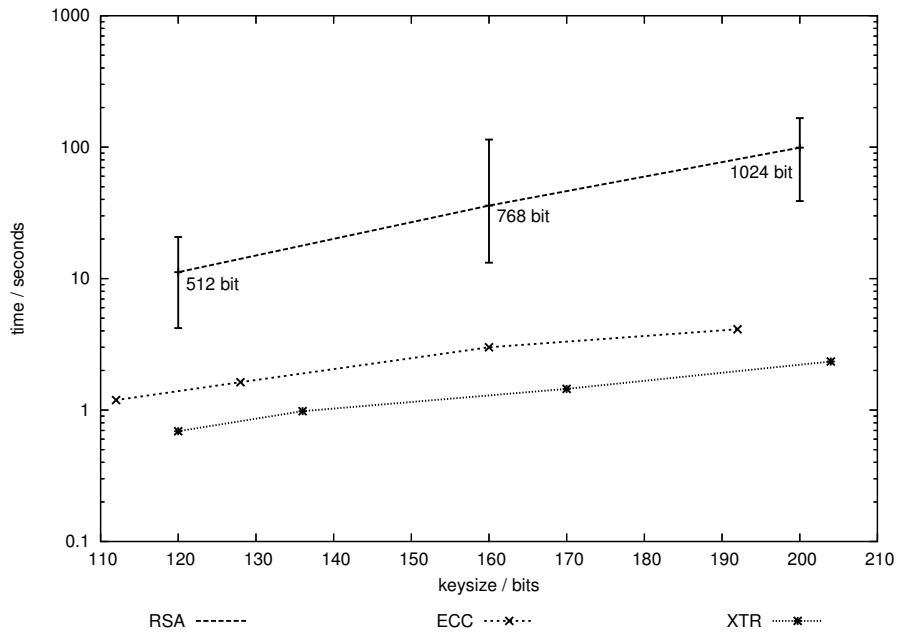


Figure 4.4: Key generation benchmark results in series 60 mobile phones

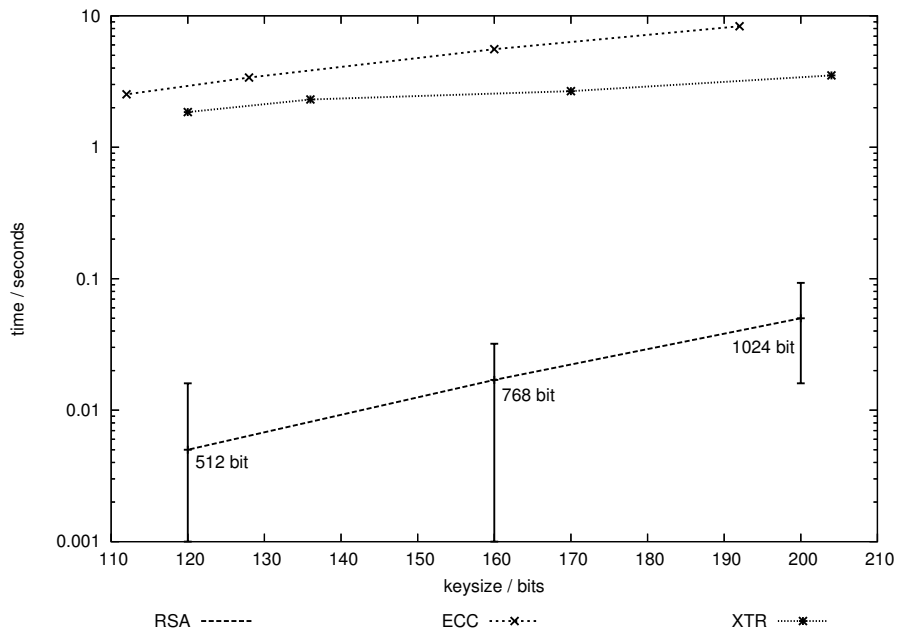


Figure 4.5: Encryption benchmark results in series 60 mobile phones

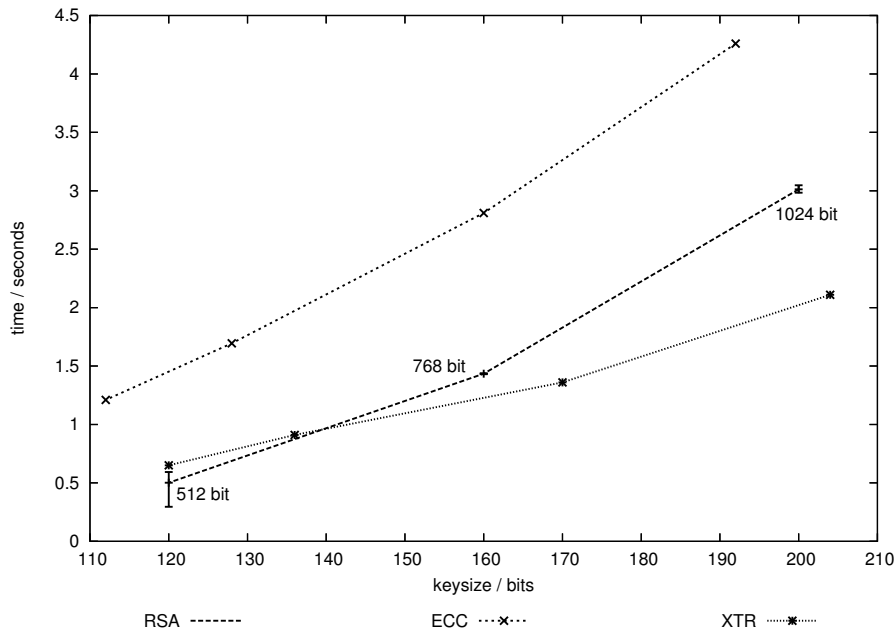


Figure 4.6: Decryption benchmark results in series 60 mobile phones

0.02 to 0.1 seconds in a series 60 mobile phone. Decryption was performed in 2.5 to 23 seconds on a series 40 mobile and in 3 seconds on a series 60 phone. Private and public key sizes are unbalanced in the used implementation of RSA. The public key is much smaller than the private key, so operations involving the public key (like encryption) take much less time than private key operations (like decryption). Decryption times on the series 60 mobile phone are very small. The Java functions used for measuring the elapsed time do not output reliable results in that time range, so the value 0 was sometimes returned.

4.1.2 ECC

ECC operations show more constant results: keypairs with 160 bit are generated in 14.2 seconds on a series 40 mobile phone and in 3.0 seconds on a series 60 device. One random number is chosen, and one multiplication in the EC is calculated in the key generation process. No trial-and-error algorithm is used like in RSA.

Encryption and decryption is performed in 30.0 seconds and 14.9 seconds respectively on a series 40 device and in 5.6 seconds and 2.8 seconds on a series 60 device. Time for ECC encryption seems to be twice the time for key generation and decryption. This is consistent with the theory: one EC multiplication is performed for key generation, two for encryption and one for decryption. Thus the dominant operation is the EC multiplication which takes in average 14.2 seconds on a series 40 device and 2.9 seconds on a series 60 mobile phone.

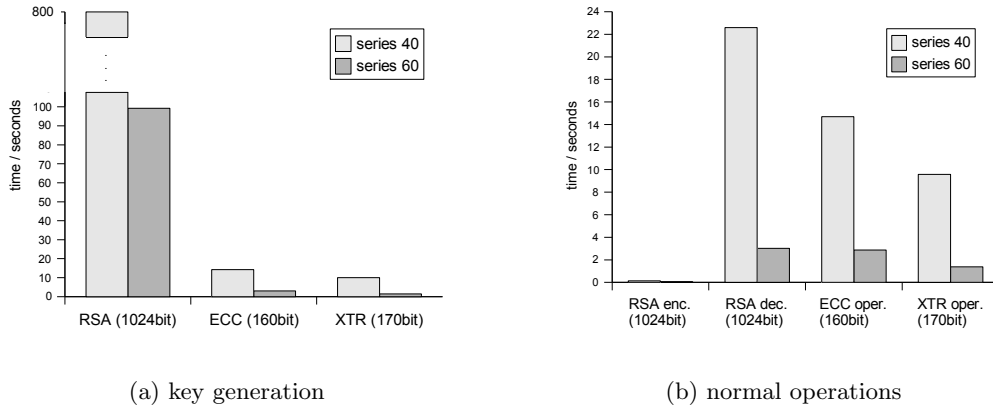


Figure 4.7: Performance comparison of RSA, ECC and XTR on series 40 and 60 mobile phones

4.1.3 XTR

Key generation of XTR keypairs with sizes of 170 bit take 10 seconds on a series 40 device and 1.5 seconds on a series 60 mobile phone. Encryption and decryption operations finish after 18.8 seconds and 9.4 seconds respectively on a series 40 mobile phone and after 2.7 seconds and 1.4 seconds respectively on a series 60 device. Exponentiations are the most computationally expensive operations in XTR, similar to the multiplications in ECC. Key generation performs one exponentiation, encryption two exponentiations and decryption one exponentiation. The calculation for one exponentiation with a keysize of 170 bit takes in average 9.6 seconds on a series 40 mobile phone and 1.4 seconds on a series 60 device.

4.1.4 Discussion

Key generation in the RSA cryptographic system takes too long on mobile devices for practical use. As displayed in figure 4.7 (a), even on series 60 mobile phones the process takes in average 100 seconds, and this value does not regard the large time variance of the calculations. Although key generation has to be performed only once, the device would not be accessible at all during that time. The RSA system's advantage is the usage of unbalanced key pairs. The time needed for encryptions using the public key is not noticeable on the devices, as can be seen in figure 4.7 (b).

The performance of ECC operations is a bit better compared to RSA decryption. This result is disappointing, as ECC is highly recommended in literature because of its vast performance advantages. For example Lauter stated in his paper [36] that ECC is 5 to 15 times faster than RSA. A reason for the bad per-

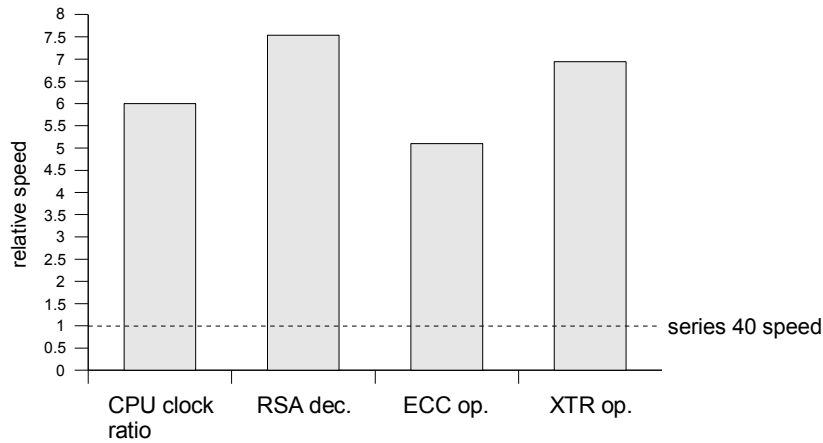


Figure 4.8: Speed of a series 60 mobile phone compared to a series 40 device

formance could be the implementation of the Bouncy Castle Cryptographic API, which might be inefficient. Key generation both by ECC and XTR take much less time than RSA, as only a random number has to be selected followed by one asymmetrical operation in the corresponding cryptographic system. Generating keys in those systems is feasible for practical applications.

XTR operations show slight performance advantages over ECC. The performance benchmark results suggest to use XTR, but as it was only introduced five years ago, minimal scientific effort has been invested into the algorithm. So the cryptographic algorithm is not trusted yet and is not in widespread use.

Both ECC and XTR are not freely available, there are patents protecting the technology. This hinders scientific research for those protocols, especially for XTR: There are few key exchange protocols available for this cryptographic system: The only ones were proposed by Lenstra in his paper [41]. His suggestions do not include a secure authenticated key exchange protocol, so XTR is not well suited for key exchange operations.

Overall, the use of ECC can be recommended for asymmetrical cryptographic operations. It is much faster in key generation than the current state-of-the-art cryptographic algorithm RSA and also shows performance improvements in decryption. Besides, already much scientific work has been spent on ECC and it is well recognized in the scientific community.

Although mobile devices still lack the speed to compute asymmetric operations without noticeable delays, technological advancements promise much higher

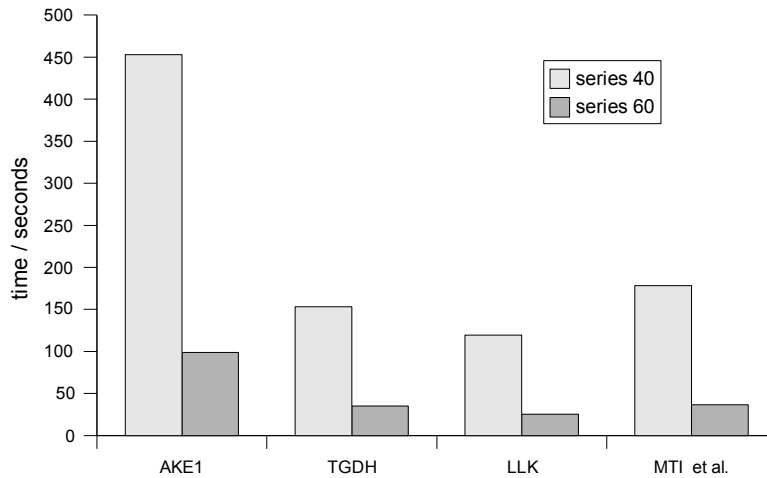


Figure 4.9: Comparison of initial setup for 5 members between series 40 and 60 mobile phones

speeds. A relative performance comparison between series 40 and 60 mobile phones can be seen in figure 4.8. In theory, a series 60 mobile phone has 6 times the speed of a series 40 device, as the clock speed of a series 60 mobile phone is 6 times higher. Of course this is only a very theoretical measure, as the overall computational speed is determined by all components used for processing and not just by the CPU clock rate. The RSA decryption operation with a key size of 1024 bits performs 7.5 times faster, ECC operations with a key size of 160 bit perform 5.1 times faster and XTR operations with 170 bit key sizes perform 7.4 times faster than on series 40 mobile phones. The different calculations involved in each of the asymmetrical cryptographic systems pose different burdens on selected components of the processing hardware, so there is a variance in the resulting speed results.

4.2 Comparison of key exchange protocols

The performance of asymmetrical cryptographic algorithms was analyzed in the last section and we recommended the usage of ECC. This section will combine the results of the theoretical analysis of secure group protocols in section 3.3 with the analysis of asymmetrical algorithms and will determine which protocol is most suitable for low-power mobile devices. Again, we will analyze the three different protocol operations initial setup, member join and member leave. For a detailed explanation, see section 3.3.

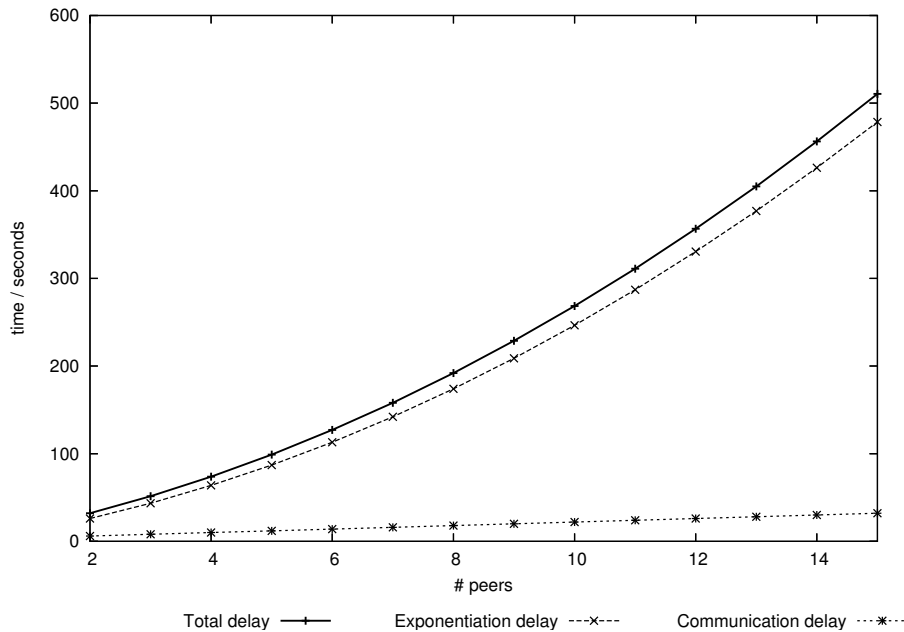


Figure 4.10: Performance of protocol AKE1 for initial setup

To illustrate the theoretical results, we assume that one asymmetrical operation takes 2.9 seconds. This is the value we obtained for ECC with 160 bit keys on a series 60 mobile in the test of asymmetrical algorithms. A series 40 device needs 14.7 seconds for one asymmetrical operation. Figure 4.9 compares the performance of a set of series 40 mobile phones and a set of series 60 mobile phones in a group of 5 devices. The fastest initial setup that can be performed by the series 40 devices takes 2 minutes, which is too much for practical applications. Series 60 mobile phones only perform the initial setup in only 25 seconds. So we will only regard series 60 devices in the remaining part of the analysis.

Network lag is assumed to be 2 seconds, as this is average in GSM data networks. The protocols only exchange very few information during the key exchange, so the available network bandwidth does not delay the message transfer. The value may differ for the used wireless protocol, but we assume it to be constant to simplify calculations.

At first we will discuss the performance of the contributory protocols AKE1 and TGDH, after that we will analyze centralized protocols where a GM manages key distribution.

4.2.1 Contributory key agreement

AKE1 needs $O(n^2)$ asymmetrical operations and $O(n)$ rounds of message transfers for initial setup. Figure 4.10 shows the results of the analysis: The time used

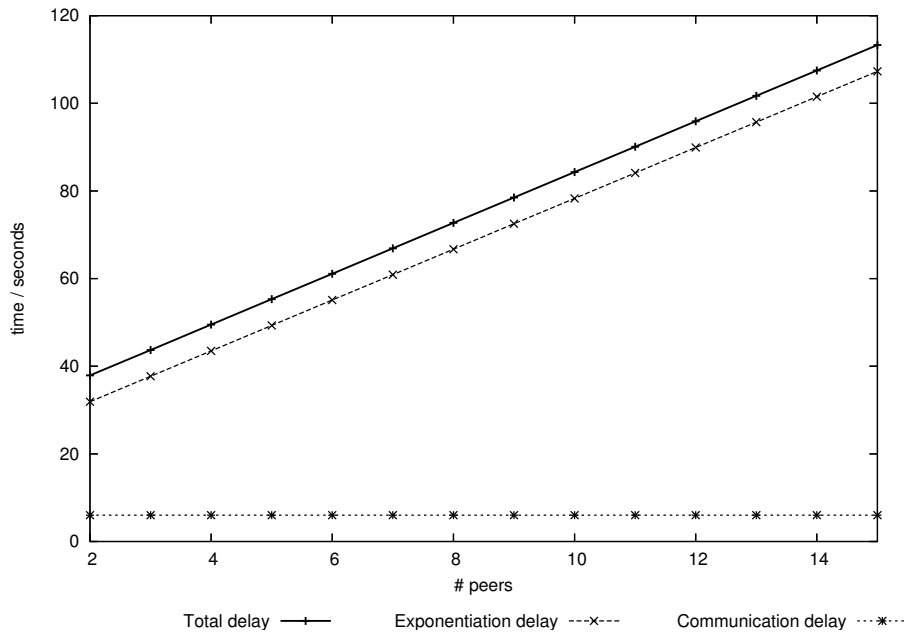


Figure 4.11: Performance of protocol AKE1 for member join

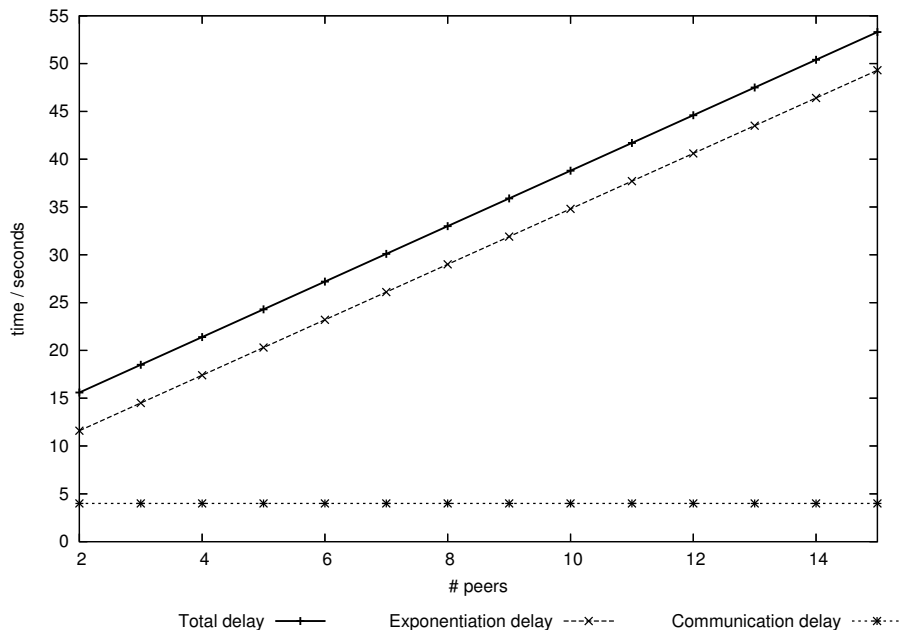


Figure 4.12: Performance of protocol AKE1 for member leave

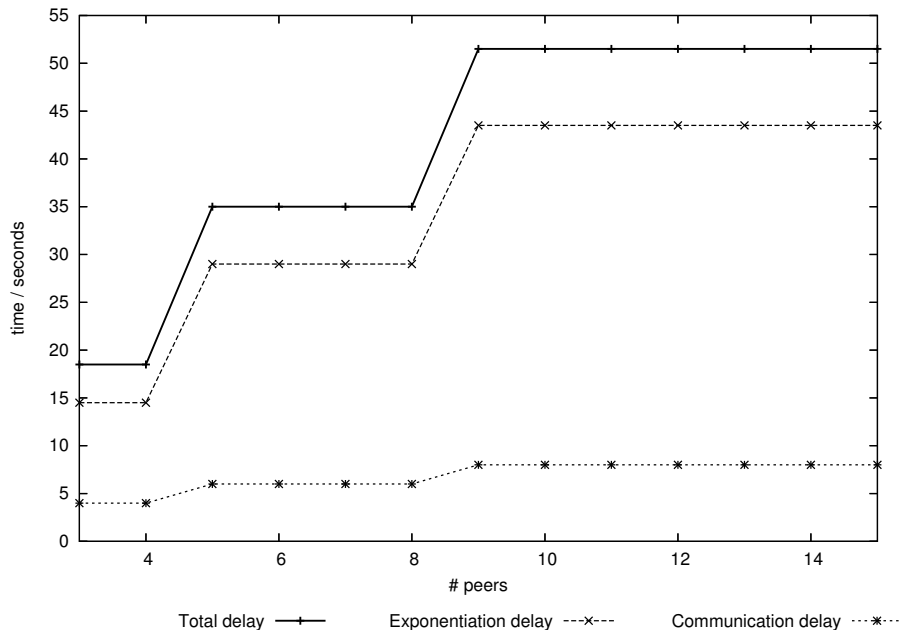


Figure 4.13: Performance of protocol TGDH for initial setup, member join and leave

for computation dominates the protocol’s execution time. For a group size of 5 people 87 seconds for computations and 12 seconds for message transfers are spent. The initial setup takes 99 seconds in total. A member join only needs $O(n)$ asymmetrical operations and 2 rounds of messages. Figure 4.11 shows the times needed, when a host joins a group of n members. A join operation with 5 devices in the original group takes 55.3 seconds in total, thereof 49.3 seconds are spent on calculations and 6 seconds are needed to transfer the messages. Member leaves take half the time for calculations compared to join operations and also 2 rounds of messages are transferred. Figure 4.12 shows the times needed when one host leaves a group consisting of n devices. In a group of 5 devices this would result in 20.3 seconds for calculations, 4 seconds for message transfers and 24.3 seconds in total.

The modified TGDH protocol needs $O(\log_2 n)$ asymmetrical operations and $O(\log_2 n)$ rounds of message transfers because of its tree-based structure. The logarithmical cost enables TGDH to be very efficient in larger groups. All operations in TGDH take the same time, as always all levels of its key tree have to be updated. There are not all devices involved in the calculations, but nevertheless the same amount of sequential message transfers and calculations is used. Figure 4.13 shows the time needed for an initial setup with 5 members, a member join with an original group of 4 members and a leave operation with an original group

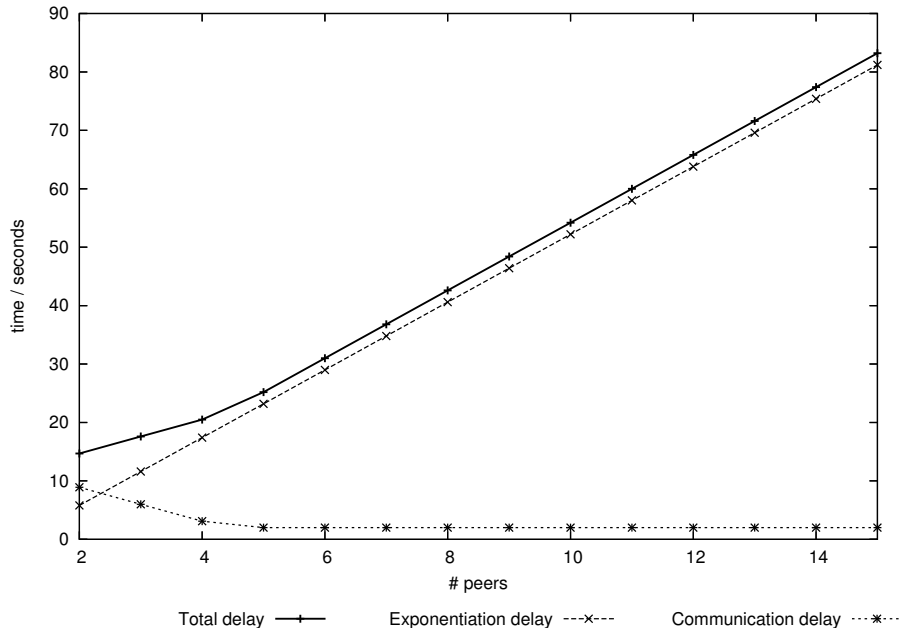


Figure 4.14: Performance of centralized multiparty protocol using LLK for initial setup

size of 6 devices. The figure shows clearly when the tree height increases, as more keys have to be computed then. The exact values for initial setup in a group with 5 devices are 29 seconds for calculations, 6 seconds for message transfers and so 35 seconds in total. A member join results in identical values, as the tree height does not change. A leave operation in a group of 5 members reduces the tree height by 1, so the times needed are 14.5 seconds for asymmetrical operations, 4 seconds for message transfers which sums up to 18.5 seconds for the whole protocol operation. TGDH exchanges many message simultaneously.

4.2.2 Centralized key agreement

The centralized group key distribution protocol which makes use of LLK uses $O(n)$ asymmetrical operations and $O(1)$ rounds of message transfers. This is expected, as the GM performs a 2-party key agreement with all the other members and each key agreement takes a constant time. Figure 4.14 shows the times for initial group setup. Idle time of the GM is included in the graph for communication delay. Communication delay starts at 9 seconds and decreases to 2 seconds with a group size of 5. After that it remains constant and is only caused by the protocol's final stage, the GEK distribution. As expected, the delay caused by asymmetrical operations is linear to the group size. The initial setup in a group with 5 hosts takes 25.2 seconds in total, whereof 23.2 seconds are needed by calculations and 2

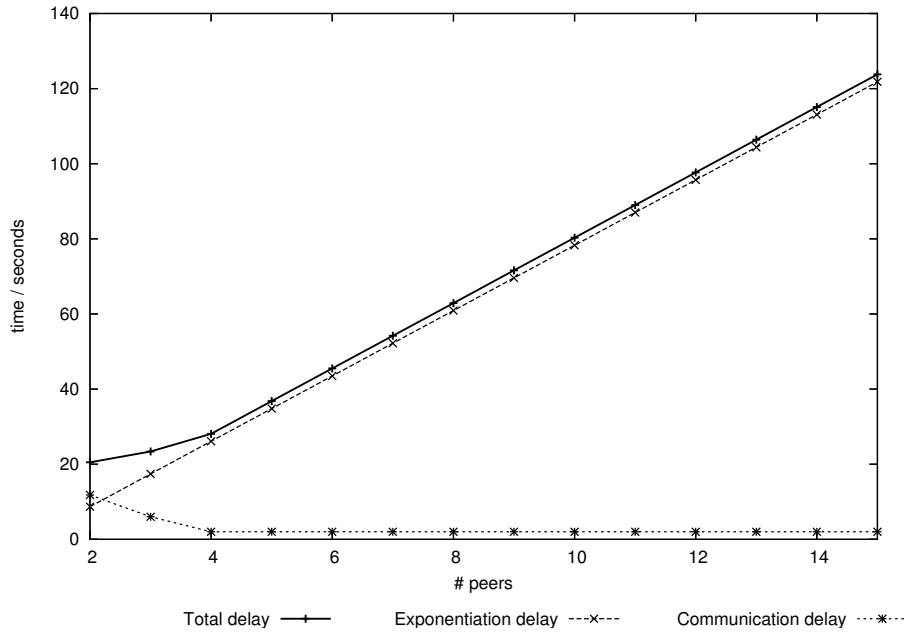


Figure 4.15: Performance of centralized multiparty protocol using a 2-party key exchange protocol similar to MTI/A0

seconds for message transfers. Member joins involve a 2-party key exchange with a following GEK distribution to all devices. Neither calculations nor message transfers are dependent on the group size, so a join operation always takes 8.7 seconds for asymmetrical operations and 6 seconds for message transfers. The whole operation takes 14.7 seconds in total. Member leave operations involve a single redistribution of the GEK, so no notable calculations are involved but a single message transfer which takes 2 seconds.

Figure 4.15 shows the results for 2-party key exchange protocols which need 3 exponentiations like MTI/A0 for setup: In a group consisting of 5 members calculation takes 34.8 seconds and message transfers 2 seconds which sums up to 36.8 seconds in total for the initial group setup. Join operations take 5 sequential asymmetric operations and thus perform in 14.5 seconds. Message transfers take 6 seconds, which is identical to LLK. In total, a member is joined after 20.5 seconds. Leave operations again only consist of a GEK distribution and thus the times do not differ from LLK: no calculations are needed and only one message transfer which takes 2 seconds.

4.2.3 Comparison

The goal of this work is to find a secure multiparty protocol for applications like video conferencing which run efficiently on mobile devices. Therefore we do

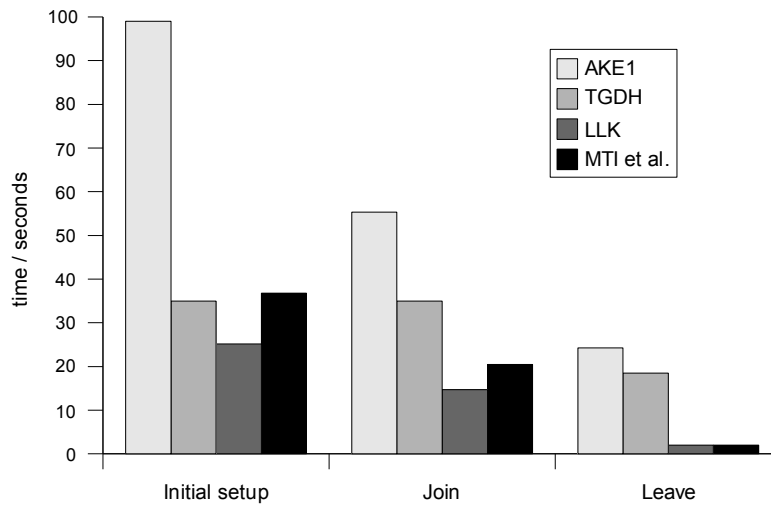


Figure 4.16: Performance comparison of multiparty protocols with 5 group members

not expect more than ten members communicating with each other. A typical situation would be 5 members or less in a group. Using this assumption, we will compare the analyzed protocols with each other and decide which one are most suitable to our requirements.

Figure 4.16 shows the total times needed for each protocol and operation. It summarizes the values which were presented in the previous section with operations for 5 group members. AKE1 is the slowest protocol in all three cases. The tree-based protocol TGDH is much faster in initial setup, but still needs twice the time for join operations compared to the centralized protocols and a member leave is more than ten times slower. MTI/A0 et al. is slower than TGDH and LLK in the initial setup, but almost twice as fast as TGDH in the join operation. Overall, LLK proves to have the best performance in all three operations.

Contributory protocols have advantages (see section 2) over centralized protocols. Those advantages are bought at an expensive price - especially AKE1's performance is much worse than the centralized protocol's performance. In each protocol operation every member of the party needs to perform asymmetrical operations, which is not the case for centralized key exchange protocols. Those have some performance advantages through the use of symmetric encryption between group manager and members. Particular leave operations are very efficient in centralized protocols: the GM chooses randomly a new GEK and the only notable

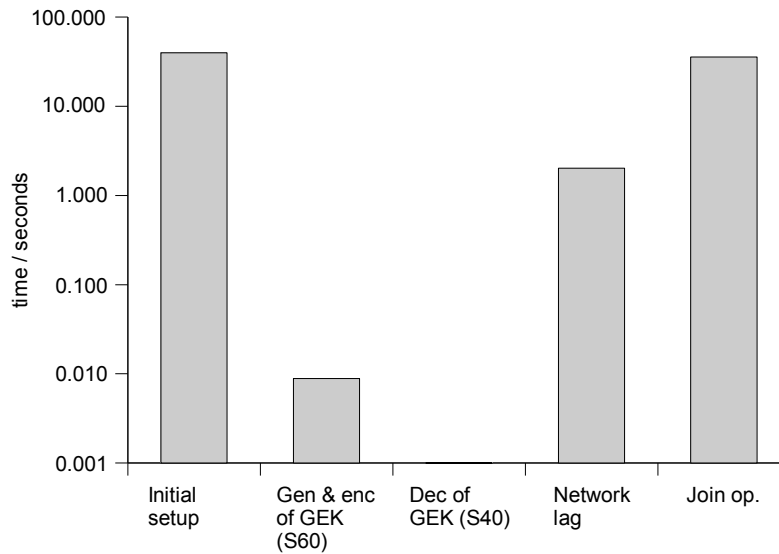


Figure 4.17: Results of the centralized key exchange benchmark with 3 mobile phones using LLK

delay is caused by the message transfer, whereas contributory protocols always have to perform computationally expensive asymmetrical operations. Unfortunately, centralized protocols are not well scalable, as the GM has to perform a linear amount of computations to the group size.

The only protocol which may become faster than LLK in larger groups is TGDH. But use of the protocol is strongly discouraged, as we used an artificially enhanced version with authentication and we strongly doubt that the protocol is secure. As a result, a centralized protocol using a fast 2-party key exchange protocol like LLK seems to perform best in the given conditions.

This analysis does not include the available network bandwidth in its calculations. If many messages are sent simultaneously to many hosts, the bandwidth may not be sufficient to send them without delay. So the stated values are lower bounds using the assumption that the network transfer delay is 2 seconds.

4.2.4 Benchmark

The results obtained by the benchmark involving one series 60 mobile phone as GM and two series 40 mobile phones as members can be seen in figure 4.17. All protocol operations were executed 8 times and the average values are shown

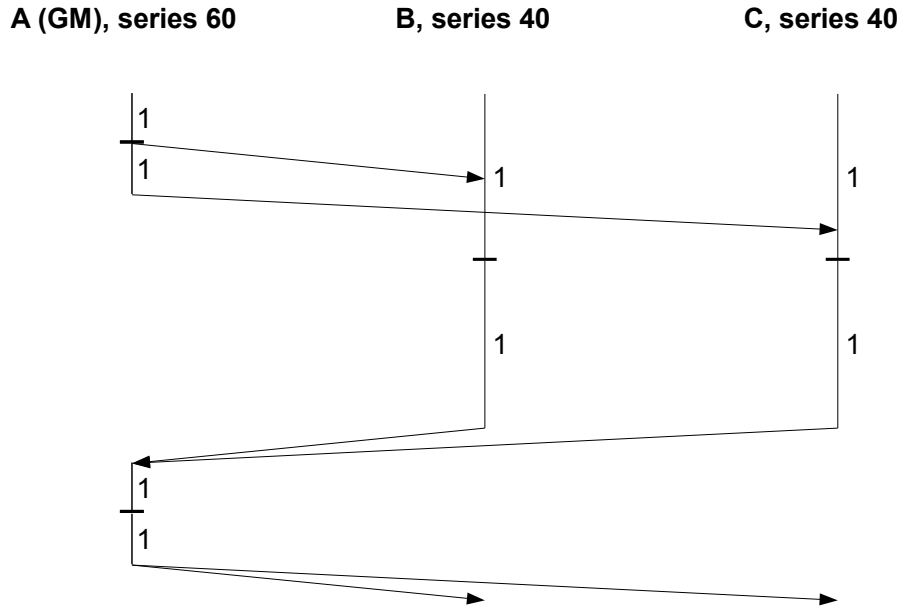


Figure 4.18: Time diagram of the benchmark's key exchange process for initial setup

here. The test performed a centralized key agreement using LLK as 2-party key exchange protocol. Certificate's signatures were not validated. The initial setup was performed as described in section 3.3.2. It took 39.7 seconds in total. A join operation needed 35.8 seconds involving the series 60 mobile phone as GM and one of the series 40 devices as joining device. Generation, symmetrical encryption and decryption of keys caused no noticeable delays: The generation and encryption of the GEK on a series 60 mobile phone took 0.009 seconds and the following decryption on a series 40 device took 0.001 seconds. A message consisting of 10 bytes sent from one device to the other arrived after 2 seconds at the receiving host, so the network lag is 2 seconds in the used wireless network system.

To confirm the validity of the theoretical key exchange analysis, we try to reproduce the results for the key exchange benchmark in theory and compare the theoretical and experimental values.

So far we have not included devices with different speeds in our analysis, so we derive them graphically using figure 4.18 which shows the time diagram of the benchmark's initial setup. Series 60 and series 40 mobile phones are assumed to take $t_e(s60) = 2.9$ seconds and $t_e(s40) = 14.7$ seconds respectively for one ECC operation. Those values are represented in scale in the figure. They are taken from the previous benchmark for asymmetric cryptographic algorithms. The network

delay was measured to be $t_m = 2$ seconds. So the total time used for an initial setup is then

$$t = 2t_e(s40) + t_m + 2t_e(s60) + t_m = 39.2 \text{ seconds}$$

This result corresponds reasonably well to the experimental benchmark result with a difference of 0.5 seconds. A join operation would take

$$t = 2t_e(s40) + t_m + t_e(s60) + t_m = 36.3 \text{ seconds}$$

The difference between theoretical and experimental result is 0.5 seconds, which also confirms the theoretical calculations.

Performance of real key exchanges will be slower, as more messages have to be exchanged between the hosts than included in the analysis. Examples would be messages to determine the group structure or to elect the GM.

Network structure from providers We determined in the benchmark tests that mobile phones are protected from the internet by the provider's firewall. Each device is given a private IP address and access to the internet is accomplished by NAT translation on the provider's firewall. Thus no connections can be established from the internet to the mobile phone, only the other direction is possible. This network structure does not cause any problems, when mobile devices want to connect to each other and are located in the same provider's network. However, if some devices are connected to different providers, the devices will not be able to connect to each other. Devices which own private IP addresses are not reachable from networks which are only connected with each other by the internet. This is a serious limitation for multiparty applications which do not rely on a trusted third party server in the internet. Hopefully this limitation will be resolved in the near future.

Chapter 5

Conclusion and Future Work

This thesis analyzed several secure multiparty protocols for their feasibility to use them in groups only consisting of mobile phones.

Especially the case of smaller groups with 10 hosts or less was considered for groupware applications like teleconferencing, video streams or shared white boards. Special attention was given to the autonomy of the group: the secure protocol should not require a trusted third party like a server to run properly. A group of mobile devices should be able to set up secure communications autonomously.

Mobile devices offer only very limited computational performance, as their main application is voice communication and they are designed to run as long as possible on battery. So fast processors which use much power are not available for applications running on those devices. Some cryptographic algorithms require a substantial amount of calculations, so it is not sure, if they are able to run in this environment.

Secure protocols require authentication methods to verify the identity of their communication peers. The most secure systems are based on public/private key cryptography. Those require asymmetrical cryptographic calculations have high computational costs. We analyzed the performance of the algorithms RSA, ECC and XTR on mobile phones. Initial key generation for RSA required too much time to be used in practical applications. ECC and XTR showed much better performance figures, as they require much smaller key sizes and thus less calculations than RSA. XTR has been introduced only a few years ago and there are only very few secure protocols available for this cryptographic system, so it is not recommendable for use yet. ECC has been tested thoroughly in the last two decades, and already many key exchange protocols are available for it. We chose ECC for the performance evaluation of secure group protocols.

We only analyzed protocols which provided authentication and made use of asymmetric cryptography. There are two different types of secure multiparty protocols: Contributory protocols do not need a central manager which controls

security and they treat every group member identically. Centralized protocols make use of a group manager which manages the protocol's security and maintains secure connection channels to every group member. We analyzed the performance of the contributory protocols AKE1 by Bresson et al. [8], a modified version of TGDH by Kim [32] and a centralized protocol using the 2-party key exchange protocol LLK by Lee [39] and using a slower protocol similar in performance to MTI/A0 [42]. We considered the protocol operations initial setup, member join and member leave.

Each case was evaluated using the results for ECC from the test of asymmetric cryptographic algorithms. Contributory protocols always perform calculations on all hosts due to their decentralized nature. Especially for member join and leave operations the centralized protocols have an advantage, as only the GM and one other host have to perform expensive calculations. Initial setup of TGDH shows promising performance for groups with more than 10 members, as the cost increases logarithmically to the group size, whereas all other protocols' cost increases at least linearly. Overall, centralized protocols perform best for applications needing a group size of 10 hosts or less. The faster the used 2-party key agreement protocol are, the better the centralized protocols perform. The fastest protocol was a centralized protocol using LLK as key agreement protocol. It took 25.2 seconds for initial setup, 14.7 for a member join and 2 seconds for a member leave with a group of size 5 hosts. Those results are valid for series 60 mobile phones. The times are small enough for practical use only for few applications, as users do not want to wait that long until communication is set up. The current state of technology does not permit this type of communication yet.

Our tests revealed, that networks of mobile phone providers use private IP addresses and are protected from the internet by a firewall. It is not possible to establish a connection between devices in two different providers' networks yet. This is a serious limitation for multiparty applications. But with the promising development of the network's bandwidth in the next generation network systems, we hope that those limitations will be abolished.

The development of the mobile devices is also promising: Series 60 mobile phones are the succeeding generation of series 40 mobile phones. Their computational performance increased by 500%. If the generation following series 60 mobile phones increases its speed by the same value and networks decrease message transfer times, secure communication groups could be set up in less than 5 seconds, which is a good value for practical application.

Future work

This thesis analyzed the speed of mobile devices based on implementations in Java. Many mobile phones of the newer series 60 generation already support applications in the programming language C++, which can be executed faster, as the code is directly executed by the processor and no virtual machine needs additional resources to interpret the code. The benchmarks could be ported to C++ to test possible performance improvements.

We only discussed secure group protocols theoretically and analyzed bottlenecks in their execution. However, the protocols were only roughly defined. An exact protocol description has to be introduced, before it can be taken for practical use: the content of transferred messages, rules if members are allowed to join the group after an invitation only or at all times, what public-key certificates may be used, etc. The goal of the work should be to develop a protocol and its corresponding API which is as easy to use as SSL for 2-party communication.

Bibliography

- [1] K. Al-Sultan et al., *A new two-pass key agreement protocol*. Proceedings of the IEEE Midwest 2003 Symp. on Circuits, Systems and Computers, 2003
- [2] Y. Amir et al., *On the Performance of Group Key Agreement Protocols*. Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems, 2002
- [3] R. Ankney et al., *The unified model*. Contribution to X9F1, 1995
- [4] G. Ateniese et al., *Authenticated Group Key Agreement and Friends*. ACM Conference on Computer and Communications Security, pp. 17-26, 1998
- [5] G. Ateniese et al., *New Multiparty Authentication Services and Key Agreement Protocols*. IEEE Journal on Selected Areas in Communications, Volume 18, No. 4, pp. 628-639, 2000
- [6] R. Blom, *An Optimal Class of Symmetric Key Generation Systems*. Proceedings of Eurocrypt '84, Lecture Notes in Computer Science, 209, Springer-Verlag, pp. 335-338, 1984
- [7] OpenSource Project, *Bouncy Castle Crypto APIs*. Website <http://www.bouncycastle.org/>, 2005
- [8] E. Bresson et al., *Provably Authenticated Group Diffie-Hellman Key Exchange*. Proceedings of ACM CCS '01, pp. 255-264, 2001
- [9] E. Bresson et al., *Provably Authenticated Group Diffie-Hellman Key Exchange - The Dynamic Case*. Source Lecture Notes In Computer Science, Vol. 2248, Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, pp. 290-309, 2001
- [10] E. Bresson et al., *Dynamic group Diffie-Hellman key exchange under standard assumptions*. In Proceedings of Eurocrypt 2002, LNCS, Springer, 2002
- [11] E. Bresson et al., *Mutual Authentication and Group Key Agreement for Low-Power Mobile Devices*. 5th IEEE MWCN, 2003
- [12] M. Burmester and Y. Desmedt, *A Secure and Efficient Conference Key Distribution System*. In Proceedings of Eurocrypt 1994, pp. 275-286, 1994
- [13] Kin-Ching Chan, S.-H.G. Chan, *Key management approaches to offer data confidentiality for secure multicast*. IEEE Network, Volume 17, Issue 5, pp. 30-39, 2003
- [14] A C-F. Chan, *Distributed symmetric key management for mobile ad hoc networks*. Proceedings of the 23. IEEE Infocom '04, pp. 2414- 2424, 2004
- [15] G. H. Chiou and W. T. Chen, *Secure Broadcasting Using the Secure Lock*. IEEE Transactions on Software Engineering, Volume 15, Number 8, pp. 929-934, 1989
- [16] D. Coppersmith and A. Shamir, *Lattice attacks on NTRU*. Advances in Cryptology - Eurocrypt '97, Lecture Notes in Computer Science, 1233, Springer-Verlag, pp.52-61, 1997

- [17] OpenSource Project, *Crypto++ Library 4.2*. Website <http://www.eskimo.com/~weidai/cryptlib.v42.html>, 2005
- [18] J. Daemen, V. Rijmen, *The Block Cipher Rijndael*. Smart Card Research and Applications, LNCS 1820, Springer-Verlag, pp. 288-296., 2000
- [19] T. Dierks and C. Allen, *Request for Comments 2246: The TLS Protocol Version 1.0*. Network Working Group, Website <http://www.ietf.org/rfc/rfc2246.txt>, 1999
- [20] W. Diffie and M.E. Hellman, *New Directions in Cryptography*. IEEE Transactions on Information Theory, Version IT-22, Number 6, pp. 644-654, 1976
- [21] Vipul Gupta, Sumit Gupta, *KSSL: Experiments in Wireless Internet Security*, Sun Microsystems, 2001
- [22] P. Gutmann, *Simplifying public key management*. Computer Publication, Volume 37, Issue 2, pp. 101-103, 2004
- [23] H. Harney and C. Muckenhirn, *RFC 2093: Group Key Management Protocol (GKMP) Specification*. Network Working Group, Website <http://www.ietf.org/rfc/rfc2093.txt>, 1997
- [24] J. Hoffstein, J. Pipher, J.H. Silverman, *NTRU: a ring based public key cryptosystem*. In Proceedings of ANTS III, Volume 1423 of LNCS, pp. 267-288, 1998
- [25] E. Hughes, *An Encrypted Key Transmission Protocol*. Presented at the rump session of crypto '94, 1994
- [26] M. S. Hwang, W. P. Yang, *Conference key distribution schemes for secure digital mobile communications*, IEEE Journal on Selected Areas of Communication, Volume 13, pp. 416-420, 1995
- [27] Min-Shiang Hwang, *Dynamic participation in a secure conference scheme for mobile communications*. IEEE Transactions on Vehicular Technology, Volume 48, Issue 50, pp. 1469-1474, 1999
- [28] A. Joux, *A one-round protocol for tripartite Diffie-Hellman*. Proceedings of Algorithmic Number Theory Symposium, Leiden, The Netherlands, pp. 385-394, 2000
- [29] M. Just and S. Vaudenay, *Authenticated Multi-Party Key Agreement*. Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology, pp. 36 - 49, 1996
- [30] Java Community Process JSR 118 Expert Group, *Mobile Information Device Profile for Java 2 Micro Edition Version 2.0*. Website <http://www.jcp.org>, 2002
- [31] B.S. Kaliski, *An Unknown Key-Share Attack on the MQV Key Agreement Protocol*. ACM Transactions on Information and System Security, Vol. 4, No. 3, pp. 275-288, 2001
- [32] Y. Kim et al., *Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups*. Proceedings of the 7th ACM conference on Computer and communications security, pp. 235 - 244, 2000
- [33] Y. Kim et al., *Tree-Based Group Key Agreement*. ACM Transactions on Information and System Security (TISSEC) archive, Volume 7 , Issue 1, pp. 60 - 96, 2004
- [34] N. Koblitz, *Elliptic Curve Cryptosystems*. Mathematics of Computation, Volume 48, pp. 203-209, 1987
- [35] H. Krawczyk, *HMQR: A High-Performance Secure Diffie-Hellman Protocol*. Cryptology ePrint Archive, Report 2005/176, 2005
- [36] K. Lauter, *The advantages of elliptic curve cryptography for wireless security*. IEEE Wireless Communications, Volume 11, Issue 1, pp. 62-67, 2004

- [37] L. Law et al., *An efficient protocol for authenticated key agreement*. Technical report CORR 98-05, University of Waterloo, 1998
- [38] P. Leadbitter and N. Smart, *Analysis of the insecurity of ECMQV with partially known nonces*. Proceedings of ISC '03, LNCS 2851, pp. 240-251, 2003
- [39] C. Lee et al., *An efficient and secure key agreement*. IEEE p1363a draft, 1998
- [40] Yu Lei, Deren Chen, Zhongding Jiang, *Generating digital signatures on mobile devices*. 18th International Conference on Advanced Information Networking and Applications, AINA 2004. Volume 2, 29-31, pp. 532-535, 2004
- [41] Arjen K. Lenstra and Eric R. Verheul, *The XTR Public Key System*. Lecture Notes in Computer Science, Volume 1880, 2000
- [42] T. Matsumoto et al., *On seeking smart public-key distribution systems*. The Transactions of the IECE of Japan, 69(2), pp. 99-106, 1986
- [43] T. Matsumoto and H. Imai, *On the key predistribution systems: A practical solution to the key distribution problem*. Advances in Cryptology - Crypto'87, LNCS vol. 293, pp. 185-193, 1988
- [44] A. Menezes et al., *Some new key agreement protocols providing mutual implicit authentication*. Workshop on Selected Areas in Cryptography (SAC '95), pp. 22-32, 1995
- [45] A. Menezes et al., *Handbook of Applied Cryptography*. CRC Press, Website <http://www.cacr.math.uwaterloo.ca/hac/>, 1996
- [46] V. S. Miller, *Use of Elliptic Curves in Cryptography*. Advances in Cryptology - CRYPTO, LNCS, Volume 218, pp. 417-426, 1985
- [47] S. Mitra, *Iolus: A Framework for Scalable Secure Multicasting*. Proceedings of the ACM SIGCOMM '97, pp. 277-288, 1997
- [48] M.J. Rao Moyer, P J.R. Rohatgi, *A survey of security issues in multicast communications*. IEEE Network, Volume 13, Issue 6, pp. 12-23, 1999
- [49] J. Nam et al., *A Weakness in the Bresson-Chevassut-Essiari-Pointcheval's Group Key Agreement Scheme for Low-Power Mobile Devices*. IEEE Communications Letters, Volume 9, Issue 5, pp.429-431, 2005
- [50] R. M. Needham and M. D. Schroeder, *Using Encryption for Authentication in Large Networks of Computers*. Communications of the ACM, v. 21, n. 12, pp. 993-999, 1978
- [51] S. L. Ng, *Comments on dynamic participation in a secure conference scheme for mobile communications*, IEEE Transactions of Vehicular Technology, Volume 50, pp. 334-335, 2001
- [52] OpenSource Project, *OpenSSL, current version 0.9.7g*. Website <http://www.openssl.org>, 2005
- [53] O. Pereira and J. J. Quisquater, *A Security Analysis of the Cliques Protocols Suites*. 14th IEEE Computer Security Foundations Workshop, 2001
- [54] C. Popescu, *A Secure Key Agreement Protocol Using Elliptic Curves*. International Journal of Computers and Applications, Volume 27, 2005
- [55] M. O. Rabin, *Digital signature and public key functions as intractable as factorization*. MIT Laboratory for Computer Science, Tech. Rep. MIT/LCS/TR-212, 1979.
- [56] S. Rafaeli and D. Hutchison, *A Survey of Key Management for Secure Group Communication*. ACM Computing Survey 35(3), pp. 309-329, 2003
- [57] Several authors of the Network Working Group, *Request for Comments 2401-2412: RFCs on topic IPsec*. Website http://www.ietf.org/rfc/rfc24**.txt, 1998

- [58] R. L. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*. Communications of the ACM archive, Volume 21, Issue 2, pp. 120-126, 1978
- [59] RSA Security, *Public Key Cryptography Standard (PKCS) #1: RSA Cryptography Standard*. Website <http://www.rsasecurity.com/rsalabs/node.asp?id=2125>, Version 2.1, 2002
- [60] A. Shamir, *Identity-based cryptosystems and signature schemes*. Proceedings of CRYPTO '84 on Advances in cryptology, pp. 47-53, 1985
- [61] N. P. Smart, *An identity based authenticated key agreement protocol based on the Weil pairing*. Electronic Letters, 38(13), pp. 630-632, 2002
- [62] B. Song and K. Kim, *Two-pass authenticated key agreement protocol with key confirmation*. Progress in Cryptology - Indocrypt 2000, LNCS 1977:237?249, 2000
- [63] M. Steiner et al., *Diffie-Hellman Key Distribution Extended to Group Communication*. Proceedings of the 3rd ACM conference on Computer and communications security, pp. 31 - 37, 1996
- [64] M. Steiner, G. Tsudik, and M. Waidner, *Key Agreement in Dynamic Peer Groups*, IEEE Transactions on Parallel and Distributed Systems, Volume 11, Number 8, pp. 769-780, 2000
- [65] M.A. Strangio, *Efficient Diffie-Hellmann two-party key agreement protocols based on elliptic curves*. SAC '05, pp. 324-331, 2005
- [66] Sun Microsystems Incorporated, *Java 2 Platform, Micro Edition*. Website <http://java.sun.com/j2me>, 2002
- [67] Symbian Ltd., *Symbian OS*. Website <http://www.symbian.com>, 2005
- [68] OpenSource project *TastePhone*. Website <http://tastephone.dev.java.net/>, 2005
- [69] Wen-Guey Tzeng, *A secure fault-tolerant conference-key agreement protocol*. IEEE Transactions on Computers, Volume 51, Issue 4, pp. 373-379, 2002
- [70] W.-H. Yang and S.-P. Shieh, *Secure key agreement for group communications*. International Journal of Network Management, Volume 11, pp. 365-374, 2001
- [71] Xun Yi, Chee Kheong Siew, C.H. Tan, Yiming Ye, *A secure conference scheme for mobile communications*. IEEE Transactions on Wireless Communications, Volume 2, Issue 6O, pp. 1168-1177, 2003
- [72] M. J. Yuan and J. Long, *Securing wireless J2ME*. IBM, website <http://www-128.ibm.com/developerworks/wireless/library/wi-secj2me.html>, 2002