EBERHARD KARLS UNIVERSITÄT TÜBINGEN University of Tübingen, UPPSALA UNIVERSITET Uppsala University

Evaluation of Authentication Algorithms for Small Devices

by

Tobias Bandh

Supervision:

Prof. Dr. Georg Carle, University of Tübingen,
Prof. Per Gunningberg, Uppsala University
Dr. Christian Rohner, Uppsala University

Herewith I declare that I've done this work on my own. External sources are marked as such.

Uppsala, 2004-03-12

Inhaltsverzeichnis

1 Introduction **1**

1.1 Objective Target 1

1.2 Structure 2

2 Mathematical Analysis of multiple Authentication Protocols **3**

2.1 Introduction 3

2.1.1 Assumptions 4

2.2 SSH authentication using Diffie-Hellman Key Exchange 6

2.3 Fiat-Shamir 8

2.4 Kerberos 10

2.5 Modified Needham-Schroeder Protocol 12

2.6 Comparison 13

2.6.1 Conclusion 14

3 Practial implementation and comparisons of the results to the theoretical results **15**

3.1 Introduction 15

3.1.1 The Limited Device 15

3.1.2 The Unlimited Device 16

3.2 Implementation of Fiat-Shamir 17

3.2.1 Preparations 17

3.2.2 Results 17

3.2.2.1 Measurements 18

3.3 Implementation of Fiat-Shamir shorter Version 20

3.3.1 Analysis 20

3.4 Implementation of a SSH like Protocol 22

3.4.1 Problems and Solutions 22

3.4.2 Measurement 22

3.5 Conclusions 24

4 Resume and forecast **25**

A Appendix - First Part **27**

A.1 RSA Public-Key-Algorithm . . . 27

A.1.1 Creating the keys . . . 27

A.1.2 Encryption and Decryption . . . 27

A.2 DES / 3DES - Encryption . . . 28

A.2.1 Encryption . . . 28

A.3 MD5 - Hashing . . . 31

B Implementation - Results Appendix **33**

B.1 Fiat-Shamir Basic Version Results . . . 33

B.2 Fiat-Shamir Short Version Results . . . 36

B.3 SSH-Measurements . . . 39

B.3.1 Analysis of the data . . . 41

C Source Code **45**

C.1 Fiat-Shamir . . . 45

C.1.1 FS-MIDlet . . . 45

C.1.2 FS-Servlet . . . 47

C.2 Fiat-Shamir Short . . . 49

C.2.1 FS-Short-MIDlet . . . 49

C.2.2 FS-Short-Servlet . . . 52

C.3 SSH . . . 55

C.3.1 SSH-MIDlet . . . 55

C.3.2 SSH-Servlet . . . 58

Bibliography **63**

Index **63**

1. Introduction

This work is a Studienarbeit for my studies at the University of Tübingen. I worked on it, at the Communications Research Group, Department for Information Technology, Uppsala University.
Today more and more devices have the ability to communicate, and to send and receive data. Having these Devices working with personal data, or critical data, security becomes very important. But these Devices are often very small and therefore limited in many ways, for example computing power or communication abilities. The aims of this work is to evaluate possibilities to use standard authentication protocols on limited Devices. We want to focus on the complexity of these protocols concerning computing power and communication. Aspects as power consumption, storage issues and others could not be examined due to the limitations of this work.
The work has its' basics in the Security Proxy Project[6] done at the Communications Research Group in 2003, it could be seen as a possible extension to it, where even the small devices need to authenticate itself to the security proxy. So that it can be prevented that a malicious device offers some service, in order to get for example the users passwords or cause damage by delivering invalid data.
A good example for such a malicious device is a faked temperature sensor in a cold room. If it constantly delivers the optimal expected temperature, although the temperature rises over the critical point, all goods are destroyed.

1.1 Objective Target

The Objective Target is to examine standard authentication algorithms in view of being possibly suitable for usage on small devices, in regards of mathematical complexity, meaning needed processor power, and communication. This works aims to evaluate several algorithms in order so get a feeling how much resources are used to fulfill the same task. Meaning to authenticate the device to the proxy. A device might be a mobile phone, a sensor, or another low processor, low memory device.

1.2 Structure

The work is structured as follows.
In the first part several known authentication protocols are analyzed and compared. Especially regarding the processor usage in form of processor cycles.
The second part deals with actually implementing two of these protocols, measuring their behavior in the real world, and compare these results to the theoretical part.

2. Mathematical Analysis of multiple Authentication Protocols

2.1 Introduction

In this first part several authentication protocols will be compared to each other. So called authentication protocols are used to convince a verifier that the claimant is in fact who he pretends to be. Authentication protocols are usually classified as follows:

- Username/Password-Authentication: The claimant authenticates himself using a username and a password, which is looked up and verified by the verifier.
- Zero-Knowledge-Algorithm: A Zero-Knowledge-Authentification works without revealing anything about the secret, beside the possession of it. For example in 1535 Nicolo Tartaglia found a way to solve third grade equations. He didn't tell anybody how he did it, but to any given equation he could deliver a solution.
- Systems using Symmetric Keys: In such a system, both claimant and verifier have a unique relation to each other. Both possess the same secret key. Which they use in a authentication protocol.
- Systems using Public Keys: Here the authentication is done based on a Public Key System. Now claimant and verifier don't share a secret key, but have their own public and private key. Whereof the public key is known by the other part of the protocol.

If a algorithm

should be used on a limited device it is considered useful how many processor cycles are needed until a entity B considers that the entity A has authenticated itself properly.

At first the algorithm is examined at a high level of abstraction. As a result there is a picture showing the number of exchanged messages, their content, and the related computations done of each of the protocol partners.

In a second step a closer look is taken at the computations executed. These are extracted and finally summed up, in order to compare the different algorithms.

The following authentication algorithms where chosen for further examination because they are typical examples for the above named classes of authentication protocols:

- SSH-Protocol using Diffie-Hellman-Key-Exchange in the first phase
- Fiat-Shamir-Authentication as an example of a Zero-Knowledge-Protocol
- Kerberos as a example for authentication using symmetric keys
- A modified Version of the Needham-Schroeder-Protocol as an example for a Challenge-Response protocol using Public Keys

2.1.1 Assumptions

In order to compare quite different protocols to each other we had to find a common abstraction level, so we had to do the evaluation on a low hardware level. Because not all protocols make use of mathematical means as for example simple multiplications. Some as for example DES or MD5 make extensive use of bitwise operations.
So each protocol was brought down to the level of processor cycles[1] or to simple multiplications, that can easily be transformed to processor cycles.
To get to consistent results we were assuming the following things:

- MD5 is used to calculate hashes
- RSA is used for public-key systems
- DES is used for symmetric key encryption

All these Methods are explained in the Appendix.

For the calculations the following assumptions are made:

- t, h, n are variables for the bitlength of the numbers
- For the modulo calculations it is assumed, that the first number has twice as much bits as the divisor
- Any exponentiation is done in a multiplicate group of order $(p-1)$ with a prime p.

[1] Reference Model was the Intel i386 processor

- K^+ means public key encryption
- K^- means public key decryption
- S is the length of a input chain, used for exponentiation
- Random numbers are only 1 cycle, because they are chosen of a precomputed list.

Calculations		Number of simple multiplications		reference
g^x	$\rightarrow$	$(t+h-2)$	$\rightarrow$	fixed base windowing
$mod\ p$	$\rightarrow$	$(2n-n)(n+3)$	$\rightarrow$	Multiple-precision division
r^2	$\rightarrow$	$\frac{1}{2}(n^2+n)$	$\rightarrow$	Multiple-precision squaring
K^+	$\rightarrow$	$B(s)$	$\rightarrow$	Addition-chain exponentiation
K^-	$\rightarrow$	$B(s)$	$\rightarrow$	Addition-chain exponentiation

These values and algorithms are taken from the *Handbook of Applied Cryptography*[3]

2.2 SSH authentication using Diffie-Hellman Key Exchange

Here we will have a closer look at the SSH-Protocol, and specially the Diffie-Hellman Key Exchange[5]. The SSH-Protocol is a typical example for a Challenge-Response Protocol, where the claimant is challenged by the verifier to respond with username and password. In a first phase the protocol uses the Diffie-Hellman-Keyexchange to establish a session key, which is used to encrypt further traffic. We assume this protocol being secure, under the assumption that there exists no effective algorithm to calculate a discrete logarithm, and not man in the middle attack only passive listening.

The Algorithm works as follows:

- Alice (A) selects a big value p, a generator value g and a Secret x with $1 \leq x \leq (p-1)$
- A calculates $u = g^x$ mod p
- A sends g,p,u to Bob (B)

- B selects a secret y
- B calculates $k = z^y$ and $v = g^y mod\ p$
- B sends v to A

- A calculates $k = v^x$

In the end both have the same secret the can use as a key k beeing:

$$k = g^{xy} \ \ mod \ \ p = (g^x \ \ mod \ \ p)^y = (g^y \ \ mod \ \ p)^x$$

The secret can be used for further secure communications e.g. to exchange authentically certificates or cryptographic keys.

Part I

	A		B	
1	$p \in$ prime	$\longrightarrow^2$	$p \in$ prime	1
1	$g \in_R \{2 \dots p-2\}$	$\longrightarrow$	$g \in_R \{2 \dots p-2\}$	1
1	$x \in_R \{2 \dots p-2\}$		$y \in_R \{2 \dots p-2\}$	1
$(t+h-2)$	$u = g^x mod\ p$	$\longrightarrow$		
		$\longleftarrow$	$v = g^y mod\ p$	$(t+h-2)$
				$+(2n-n)(n+3)$
$\underline{\underline{t+h-2}}$	$k = (g^y)^x mod\ n$		$k = (g^x)^y mod\ n$	$\underline{\underline{t+h-2}}$
n^2+7n+7		total multiplications		n^2+7n+7

Now it is possible to just sum up the simple multiplications for each side:

A $1+1+1+(n+n-2)+(2n-n)(n+3)+(n+n-2) = \underline{\underline{n^2+7n+7}}$

B: $1+1+1+(n+n-2)+(2n-n)(n+3)+(n+n-2) = \underline{\underline{n^2+7n+7}}$

At this point A is not yet authenticated to B. There is additional computing for to precompute the prime, and of course the second part has to be taken into this consideration. What is quite astonishing at this point, is that A and B have to do the same amount of computations.
In order to get a feeling for the size of the used numbers we had a quick look at a actual implementation of the SSH Protocol[1] where we could see that it uses $q=3$ and p some 128Bit prime.

Part II - Authentication

	A		B	
$b_{des}(748+10s)$	$K^+_{AB}(login)$	$\longrightarrow$	$K^-_{AB}(login)$	$b_{des}(748+10s)$
$b_{des}(748+10s)$	$K^-_{AB}(password?)$	$\longleftarrow$	$K^+_{AB}(password?)$	$b_{des}(748+10s)$
$b_{des}(748+10s)$	$K^+_{AB}(password)$	$\longrightarrow$	$K^-_{AB}(password)$	$b_{des}(748+10s)$
			pass ok?	1

2.3 Fiat-Shamir

The basic Fiat-Shamir[2] authentication protocol is a Zero-Knowledge protocol. The objective of this protocol is that A shows that she owns a secret, without sending the secret or parts or any information about it over the net. The protocol goes over multiple rounds, until the verifier is convinced of As' identity. It is assumed that the protocol is complete and sound, in regard to the definitions given in the 'Handbook of Applied Cryptography' [3]. This protocol is based on the assumption that some computations are easy to do, but the way back is quite hard. In Order to get a independent and reliable number n, there is a so called Trust-Center that takes care of the calculation of n.

The Algorithm can be divided into two Parts:

- One-Time-Setup
  - A Trust-Center publishes a product of two primes $n = p \cdot q$
  - A Client A chooses a secret s which is a coprime to n: $1 \leq s \leq (n-1)$
    With this coprime A computes her public key, that is registered at the Trustcenter. $v = s^2 \ mod \ n$
- Protocol Actions
  - A chooses a random commitment r: $1 \leq r \leq n-1$
  - A calculates $x = r^2 mod \ n$ and send it to B
  - B selects randomly a challenge $e \in [0,1]$ and sends this to A
  - A: if challenge$e = 0 \rightarrow$ A sends $y = r$ to B
    if challenge $e = 1 \rightarrow A$ sends $y = r \cdot smod \ n$ to B
  - B rejects the prove if $y = 0$
    B accepts the prove if $y^2 \equiv xv^e$

A has selected her secret s, calculated the public key and registered it at the Trust Center.

	A		B	
1	$s: \quad 1 \le s \le (n-1)$			
	$coprime(n, s)$			
1	Public Key[3]: $v = s^2 \ mod \ n$			
1	$r \in [1, (n-1)]$			
$\frac{1}{2}(n^2+n)$	$x = r^2 \ mod \ n$	$\longrightarrow$		
$+(2n-n)(n+3)$				
		$\longleftarrow$	$e \quad \in \quad [0, 1]$	1
0	$e = 0 : y = r$	$\longrightarrow$		
(n-1)(n-1)	$e = 1 : y = r \ \cdot \ s \ mod \ n$	$\longrightarrow$	$y = 0$: reject	
+(2n-n)(n+3)			$y^2 \equiv x \cdot v^e$: accept	$\frac{1}{2}(n^2+n)$
				$+(2n-n)(n+3)$

Now again the number of simple multiplications can be summed up. Because of the probability getting a one or zero as repsonse causes the need of a probality b which is, because of the random out of two elements b= $\frac{1}{2}$.

A: $3 + (\frac{1}{2}(n^2+n) + (2n-n)(n+3) + \frac{1}{2}[(n-1)(n-1) + (2n-n)(n+3)]$
$= \underline{3 + \frac{1}{2}(5n^2 + 2n + 1)}$

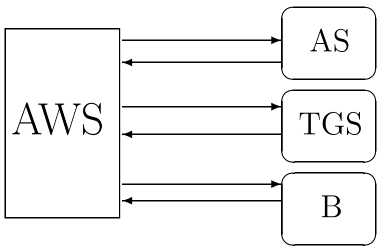
B:$1 + \frac{1}{2}[(n-1)(n-1)] + \frac{1}{2}[(2n-n)(n+3)] + (\frac{1}{2}(n^2+n) = \underline{1 + \frac{1}{2}(3n^2 + 6n + 1)}$

In every round the chance of being lucky with guessing the corresponding response r without the knowledge of s is $\frac{1}{2}$. This means that the probability to successful guess in every round and to cheat the authentication is $(\frac{1}{2})^n$. In order to satisfy the verifiers demands for a successful authentication, the algorithm works with multiple rounds. To achieve a equivalent confidence as for example we would get with SSH we assume that about 20 to 40 rounds are required. The hope in using this protocol is to be able to adapt the complexity to the actual needs.
Running the protocol over x rounds does not mean that the number of messages is increased to: $number\_of\_messages \cdot \ x$, as we will show later on.

[3]It's only one calculation because it's only done once

2.4 Kerberos

Kerberos[4][p. 472 ff]/[8][p. 402 ff] is the example for a symmetric key based challenge-response-authentication protocol based on the Needham-Schroeder authentication protocol. Kerberos makes use of two services to grant access. The first is the Authentication Server (AS) which is used to authenticate a user and to grant him access to the his Workstation (AWS). The second Service is the Ticket Granting Server (TGS). The TGS hands out tickets to authenticated users (or to their machines). With these tickets the authentication towards other resources is done. Here no keys have to be shared. All are created dynamically.



The three steps to logon and access a service, are shown here:

Part I:

- Userlogon to the system
  - A sends her LOGIN to AWS
  - AWS sends $LOGIN_A$ to the AS
  - AS looks up the corresponding password hash and uses it as a key for encryption
  - AS sends $K_{A,AS}(K_{A,TGS}, K_{AS,TGS}(A, K_{A,TGS}))$ to A
    $K_{AS,TGS}(A, K_{A,TGS})$ is called a ticket.
  - After reception of AS message, AWS querys A for her password
  - A enters her password
  - AWS computes the key $K_{A,AS}$ and decrypts the message.
    If the decryption is a success, A is logged on to the network.

  Part II:

- Further authentication of the user to any available service
  If A wants to access A service offered by B, she needs to authenticate herself to B.
  - A sends the ticket $K_{AS,TGS}(A, K_{A,TGS})$, Bs' identity, and a encrypted timestamp $K_{A,TGS}(t)$ to the TGS. The timestamp is used to avoid the replay of the message.
  - TGS returns $K_{A,TGS}(K_{A,B}, K_{B,TGS}(A, K_{A,B}))$ to A, after it has decrypted the message, checked the timestamp and made shure that the user is authorized for that service.

Part III:

- As soon as A owns the new ticket, she can authenticate herself to B and use the requested service.

Defining some Messages
$(1) = A$
$(2) = K_{A,AS}(K_{A,TGS}, K_{As,TGS}(A, K_{A,TGS}))$
$(3) = A, B, N_A$
$(4) = K_{As,TGS}(A, K_{A,TGS})$
$(5) = K_{A,TGS}(K_{A,B}, N_A, L, B, K_{TGS,B}(K_{A,B}, A, L))$
$(6) = K_{TGS,B}(K_{A,B}, A, L)$
$(7) = K_{A,B}(A, T_A)$
$(8) = K_{A,B}(T_A)$

Part I

	A/AWS		AS	
	(1)	$\rightarrow$	Lookup $K_{A,AS}$	1
	(2)	$\leftarrow$	(2)	$7 * (748 + 10s)$
976	$Calc\ \ K_{A,AS}$			
$4 \cdot (748 + 10s)$	$K^{-}_{A,AS}(2)$			

Part II

	A/AWS		TGS	
1	(3)	$\rightarrow$		
	(4)	$\rightarrow$	$K^{-}(4)$	$3 \cdot (748 + 10s)$
$12 \cdot (748 + 10s)$	$K^{-}(5)$	$\leftarrow$	(5)	$8 \cdot (748 + 10s)$

Part III

	A/AWS		B	
	(6)	$\rightarrow$	$K^{-}(6)$	$4 \cdot (748 + 10s)$
$3 \cdot (748 + 10s)$	(7)	$\rightarrow$	$K^{-}(7)$	$3 \cdot (748 + 10s)$
$(748 + 10s)$	$K^{-}(8)$	$\leftarrow$	(8)	$(748 + 10s)$

2.5 Modified Needham-Schroeder Protocol

Modified Needham-Schroeder Protocol[4] is a challenge-response authentication algorithm using a public key system. For such a algorithm it is evidently that all communication partners have access to the public-key of the others. Ether exchanged in the very beginning or via a Key-Distribution-Center (KDC).

The algorithm works as follows:

- A sends her identity and a nonce[4] N_A encrypted with Bs' public key to B: $K_B^+(A, N_A)$
- B decrypts the message, and sends As' nonce, another nonce and a session key K_{AB}, encrypted with As' public key to A[5].
- A returns Bs' nonce encrypted with the session key.

After these three messages A and B have authenticated each other. To get further information on computations needed for Encryption / Decryption see Appendix 1. But to understand this table a few additional information is given:

- The b stands for the number of blocks which are needed
- b(s) means number of blocks times number of cycles needed for the task
- (748 + 10s) is the number of cycles for DES encryption
  it is 10s because of the 10 s-boxes where the number of cycles for the single S-Box is represented by the variable s (which will be replaced by a number later on)

	A		B	
$b_1(s)$	$K_B^+(A, N_A)$	$\longrightarrow$	$K_B^-(N_A)$	$b_1(s)$
$b_2(s)$	$K_A^-(N_A)$	$\longleftarrow$	$K_A^+(N_A, N_B, K_{A,B})$	$b_2(s)$
$b_{DES}(748 + 10s)$	$K_{A,B}(N_B)$	$\longrightarrow$	$K_{A,B}(N_B)$	$b_{DES}(748 + 10s)$

In order to get the number of simple multiplications for that algorithm, It is now possible to sum up. But there are three things that have to be taken into consideration first. The used encrpytion algorithms work on fixed length Bitstrings. To get these Bitstrings the message is padded to a multiple of its' needed blocklength. After that this padded message is splitet into b bitstrings: $b = \frac{PaddedMessage}{NeededBitlength}$. So the encryption has to be executed B times in order to encrypt the whole message.
As the private key is much longer than the public key, the input chains have different length. The third important thing is, that DES does not work with simple multiplications but is calculated in processor cycles. That means that $b_{DES}(748 + 10s)$ is the

[4]A nonce is a random number used to connect multiple messages together

[5]Where K_A^+ an operation with As' public and K_A^- private key is. $K_{A,B}$ is a symmetric Key for A and B

number of processor cycles for a DES encryption. To get the whole algorithm into a one sum, the public-key Part has also to be calculatet in processor cycles.

$$Cycles = 15 \cdot [b_1(s_{pu}) + b_1(s_{pr}) + b_2(s_{pu}) + b_2(s_{pr})] + 2b_{DES}(748 + 10s)$$

2.6 Comparison

After having the algorithms analyzed it should be now possible to make a comparison.
But again it is important to state some assumptions.

- DES-Keylength: 56 Bits
- Length of nonces, timestamps, lifetimes: 32 Bits
- Identitys, passwords: 128 Bits
- Fiat-Shamir over 40 rounds
- Fiat-Shamir with 32Bit numbers
- Cycles means the number of needed processor cycles (comp. Appendix)
- 1024 Bit private key K with #1(K) = 512
- 256 Bit public key PK with #1(PK) = 122

Protocol	Messages	Claimant Cycles A	Verifier Cycles B
SSH (DH + user /password)[6]	7 (5)	17887	17887
Fiat-Shamir (One Round)	4	38902	24502
Fiat-Shamir (40 Rounds)	160	1583530	490040
Kerberos (Logon to the System)	2	7968	6992
Kerberos(Logon and authentication to a Service)[7]	6	35960	45448
Modified Needham-Schroeder	3	21113	21113

[6] If you don't take the setup-messages into account, or combine the first three messages, only five messages are exchanged

[7] B is the sum of the cycles needet at AS, TGS and B

2.6.1 Conclusion

Having these numbers leads to some conclusions. Looking only at processor cycles authentification via Kerberos or SSH seems to be quite good. In order to choose the best algorithm, it might be necessary to take more things into consideration. For example the number of messages, or only the amount of data being exchanged within the authentication process.
Having a closer look at the table there are some more things to notice.

- A seems to have the bigger workload to accomplish. Only once in Kerberos B seem to have more to do, but thats not really true, because the 6992 cycles are shared by at least to ”subjects”
- All the protocols who are making use of symmetric cipher methods and therefore use of bitwise operations need less cycles than those using mostly public key methods.
- The usage of the exponentiation algorithm which uses less operations than the squaring algorithm can be identified through the smaller number of cycles in total.
- The question could be raised if setup messages and calculations should be counted or not. I think that depends on if you have to do the calculation for each time you use the protocol, or if you do it once and for example register the value at a Trust Center or similar.

3. Practial implementation and comparisons of the results to the theoretical results

3.1 Introduction

After having finished the theoretical analyzes of authentication protocols, the next logical step was to try to implement at least one or two of the analyzed algorithms. Especially to see if the theoretical results really do matter and behave like foreseen in the real world.
Instead of using any special device, the decision was taken to use a regular Nokia 6610 mobile phone.

Not all protocols were implemented, implementations of Fiat-Shamir and a SSH-like protocol using Diffie-Hellman Keyexchange. All sources are available in the Appendix.

3.1.1 The Limited Device

The Nokia 6610 mobile phone was chosen because it comes with builtin Java support. Meaning there is the possibility to download an execute Java programs that follow the MIDP 1.0 standard. In addition to that it supports connections to the Internet via GPRS. Which seemed to be great for this kind of experiment.
After some research it turned out that the phone supports only the Nokia Series 40 Developer Platform. Phones of this serie have no native Socket support so all communication has to be done using HTTP-connections. This first seemed to be a setback.

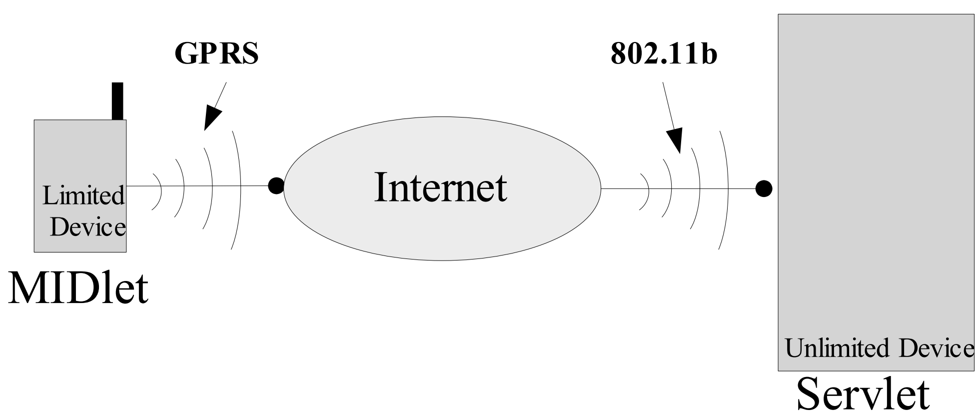
After having a closer look at a introductive paper provided by Nokia[7] the conclusion was drawn to use Apache Tomcat with Java Servlets on the server side. This allows using the same programming language on each side and eliminates the need of native socket support.

The java programs, called MIDlets might be up to 64 KByte large, they can access up to 200 KByte Heap storage, and up to 625 KByte shared memory.
The J2ME MIDP 1.0 is a very stripped down version of the Java Programming environment. There is no support for numbers larger than 64 Bit (long), also a lot of common classes as for example String Tokenizer are not available. This makes it essential to program in a efficient way, especially in respect to the memory usage. But there will be no other choice than reimplementing some needed classes.
The mobile phone is equipped with a Comviq[1] PrePaid Card and uses the Comviq GPRS System to connect to the Internet. Before any communication the phone is reset, to start all measurements with the same conditions.

Before testing the programs on the phone a simulator is used to test if they work without errors. There is another advantage, a big number of tests could be run in shorter time, and without any costs.

3.1.2 The Unlimited Device

In contradiction to the mobile phone the server is called the unlimited Device, because if you compare its possibilities to the possibilities on the phone the server has unlimited resources in storage, memory, bandwidth and processor power.
HTTP connections which are used for the communication are request-response connections which do not keep any context after delivering the response. By using Servlets it is possible to keep the context by saving session data in session variables.
Most protocols need more than one communication step. To fit them into the request-response scheme they have to be structured in phases. The current state of the protocols is saved into the session variables.
The server is connected to the Internet over a 802.11b link. All measurements were made at low traffic times on the Wireless link. So that all this does not influence the communication delay in any way.



[1]www.comviq.se

3.2 Implementation of Fiat-Shamir

The decision for Fiat-Shamir authentication as a first algorithm to implement was quite fast taken because Fiat-Shamir does not require any additional libraries, as for example a cryptographic library. Fiat-Shamir uses only multiplication and exponentiation on the limited device.

3.2.1 Preparations

In contradiction to the theoretical work it is important to make sure that the calculations at no point of time exceeds the domain of long values.
So p and q are chosen, in a way to satisfy:

$$n^2 = (p \cdot q)^2 <= 2^{63}$$

This makes sure, that all calculations resolve in numbers smaller than the maximum possible.
For the communication the messageformat is defined as follows[2]:

	A		B	
Commitment	$v = (s\_l\_v)\&x = (s\_l\_v)$	$\longrightarrow$		
		$\longleftarrow$	$(1 \mid 0)$	Challenge
Response	$y = (s\_l\_v)$	$\longrightarrow$		
		$\longleftarrow$	$(1 \mid 11 \mid 10)$	Status

If the protocol goes over multiple rounds, and the status is 1, this round succeeded, if the status is 11 the whole authentication succeeded, otherwise it might be 10, meaning that the authentication failed.

There are two possible phases of the program, in the first the servlet receives key, and commitment and returns a challenge.
In the second phase the server verifies the response at returns the status of the protocol.

3.2.2 Results

After having successfully implemented both MIDlet and the Servlet, multiple tests were run using a mobile-phone-simulator to discover and eliminate possible bugs. After the functionality of the implementation had been verified the MIDlet was downloaded to the mobile phone. We expected differences especially in regards to slower processor, lower memory, and a much smaller bandwidth, than on the simulator.

The first surprise appeared, while having a closer look at a captured communication session between the mobile phone and the server. There was a lot more of information sent, that did not appear in the simulators network monitor. Which is quite clear but was, unexpected at this point.

[2]Commitment: s_l_v = some_long_value

```
POST /ssh/ HTTP/1.1 Host: sec-proxy.no-ip.com:18080
X-Wap-Profile: ”http://nds.nokia.com/uaprof/N6610r200.xml”, ”1-4csgll//4OzaR6U7yLPStg==”, ”2-XnrTOLDzBJdZHN2vSasoNA==”,
”3-Z+lJzsFTo4qT8wYH1WfzIQ==”, ”4-pHUB9AHDokQcbAwHU9nTBQ==”

Content-Type: text/plain Accept: */* Accept-Language: de
User-Agent: Nokia6610/1.0 (4.18) Profile/MIDP-1.0
Configuration/CLDC-1.0 Accept-Charset: * Accept-Encoding: deflate, gzip TE: deflate, gzip
X-Wap-Profile-Diff: 1;
<?xml version=”1.0”?>
<rdf:RDF xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”xmlns:prf=”http://www.wapforum.org/UAPROF/ccppschema-
19991014#”>
<!–browser vendor site: Default description of properties –>
<rdf:Description><prf:CcppAccept><rdf:Bag><rdf:li>*/*</rdf:li></rdf:Bag></prf:CcppAccept></rdf:Description>
</rdf:RDF>
X-Wap-Profile-Diff: 2;
<?xml version=”1.0”?>
<rdf:RDF xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”xmlns:prf=”http://www.wapforum.org/UAPROF/ccppschema-
19991014#”>
<!–browser vendor site: Default description of properties –>
<rdf:Description><prf:CcppAccept-Charset><rdf:Bag><rdf:li>*</rdf:li></rdf:Bag></prf:CcppAccept-Charset></rdf:Description></rdf:RDF>


X-Wap-Profile-Diff: 3;
<?xml version=”1.0”?>
<rdf:RDF xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”xmlns:prf=”http://www.wapforum.org/UAPROF/ccppschema-
19991014#”>
<!– browser vendor site: Default description of properties –>
<rdf:Description><prf:CcppAccept-Encoding><rdf:Bag><rdf:li>deflate, gzip</rdf:li></rdf:Bag></prf:CcppAccept-Encoding></rdf:Description>
</rdf:RDF>
X-Wap-Profile-Diff: 4;
<?xml version=”1.0”?>
<rdf:RDF xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”xmlns:prf=”http://www.wapforum.org/UAPROF/ccppschema-
19991014#”>
<!– browser vendor site: Default description of properties –>
<rdf:Description><prf:CcppAccept-Language><rdf:Seq><rdf:li>de</rdf:li></rdf:Seq></prf:CcppAccept-Language></rdf:Description>
</rdf:RDF>
Cookie: $Version=0;wtls-security-level=none Content-Length: 39

v=1597229406&x=534021128&n=2127342101&r=1769446844
```


All the information about the device, possible protocols, the used browser, and supported languages is exchanged for each communication step. The ration between transmitted data and the actual payload is quite bad, what might influence the time needed for communication.

3.2.2.1 Measurements

In order to verify the results of the theoretical work timers were implemented to gain information on how long the steps need to be completed. Several tests were run, to avoid extreme values which might falsificate the results.
First the protocol was run over one round, followed by runs over five and ten rounds. Again each version is done several times to make sure that not only extreme values are taken into consideration. The following values are recorded. They are shown in the appendix.

- Round: Number of Rounds
- Random: Time to calculate a random number in ms
- X: Time to calculate the x-value
- Y: Time to calculate the y-value
- Challenge: The challenge sent to the limited device
- Overall: Time to accomplish the whole protocol

Analyzing the values leads to the following conclusions:

- All calculations are done faster than expected
- The protocol needs a certain startup time to establish all communication channels
- The communication delay is the main factor for the time a single round needs
- The time a round needs to finish varies

The differences observed in startup- and round-times influences the times for the whole protocol. These variations are caused by the time needed to establish the needed communication channels. After this could not be changed, we where thinking of some other ways to optimize the protocol. As it seemed that the amount of transferred data does not influence the communication time so much, but the number of messages exchanged influences the protocol-time strong, we decided to set up a second version of the protocol we will present in the next section.

3.3 Implementation of Fiat-Shamir shorter Version

In order to get a better ratio between data that is submitted and the the protocols payload, the decision was taken to implement a second version of the Fiat-Shamir-Protocol, which is adapted to submit the values for multiple rounds in one communication phase.
For example: The small device sends not only one but ten commitments, and receives therefore ten challenges. This optimizes the ratio, but also increases the need for memory to save vectors or arrays of values, which are needed for the calculation in one of the following steps. On the Serverside this causes no problems, but on the limited device it seems to do so: Due to the lack of memory it is not possible to create more than 30 commitments and store them and the associated random numbers for the next phase.
To separate the different values in the Challenge-string the StringTokenizer Class, not included in the J2ME, had to be reimplemented.

The message format had to be slightly changed to fit the new situation.
Instead of sending:

$$v = (s\_l\_v)\&x = (s\_l\_v)$$

now the message format is:

$$v = (s\_l\_v)\&x_0 = (s\_l\_v)\&\ldots\&x_n = (s\_l\_v)$$

	A		B	
Commitment	$v = (s\_l\_v)\&(x_n = (s\_l\_v))^+$	$\longrightarrow$		
		$\longleftarrow$	$((1\|0)2)^+$	Challenge
i Response	$(y_n = (s\_l\_v))^+$	$\longrightarrow$		
		$\longleftarrow$	$(1\|11\|10)$	Status

Also the responses are changed. The first response is a string of challenges separated by "2". A possible response could look like this:

12020202021212121212

After receiving it, it has to be tokenized and each challenge has to be worked on separately. Now the second request will look like that:

$$u_0 = (s\_l\_v)\&\ldots u_n = (s\_l\_v)$$

As a response to that request there are the same possibilities as in the first Version of the Fiat-Shamir-Protocol.

3.3.1 Analysis

After analyzing the new times, it was obvious that the maintenance of the values in the memory takes more time, the more values are stored in the memory. While dealing with only one value, the creation and storing of that value takes in average two to three milliseconds. But when the number of values increases to 30 Values it takes in average twice as long and in extreme cases even six times longer than before. Looking at the times the rounds need to run, there is no real trend. There is not real trend noticeable looking at the duration of a round and message length. It seems more likely that the communication delay has big variations which makes it impossible to say something about the time the protocol needs to finish. But what it is clear, that the first round takes a little longer because the communication channel has first to be established, although the channel is broken down after every round, this initiation time appears only at the first round.

- Calculations still done very quickly
- Some time is spent on handling the values
- Longer messages do not lead to longer communication time
- Communication times show still differences from round to round

This leads to the conclusion, that the Limited Device is not as limited as it was supposed to be. How ever memory shortage and communication delays should be more taken care of than on processor power.

3.4 Implementation of a SSH like Protocol

The following protocol uses in a first phase the Diffie-Hellman Key Exchange algorithm to agree on a secret shared key which is afterwards used to encrypt username and password using DES. After the the secret is successfully established the small Device builds up a request string, which is encrypted and sent to the server.The string has some similarities with a HTTP-GET-Request.

user="username "&pass="password "

On the server the string is decrypted, using the secret and user and password are verified. There are two possible replies to the authentication request: "OK"or "DENIED". The reply is encrypted and returned to the small device, which after decryption checks, wether the authentication succeeded or failed.

3.4.1 Problems and Solutions

Even before the implementation it was several problems were known, which could not be solved with the available resources. As there is no cryptographic support in J2ME the need of a suitable library, providing classes for DES-ecryption was quite obvious.
But not only the cryptographic part caused a problem. Also the calculation of the secret was not that easy. Either you could make sure that all calculations run within the range of the Long type or add some support for BigInteger. Staying within the borders of the Long Type is quite hard, because you have to calculate logarithms. And the security of the protocol decreases very strong.
The Bouncycastle Lightweight Crypto API[3] brings both, as well slasses for DES encryption and BigInteger support. This API is specially created for the use together with J2ME. It seemed to be the optimal supplement to the J2ME, solving all problems at once. Especially including the BigInteger support brought the secret into regions where it is already hard to recalculate or guess it.
While the testing phase, after the implementation, an inexplicable error appeared. About 30% of the calculated secrets were wrong. It was tracked to a point where it was clear that there had to be a Bug in BouncycastlesB́igInteger class. This was approved via the mailinglist and quite fast fixed.

3.4.2 Measurement

This time there were more measurements taken. As well in the simulator as on the phone. There were also more times taken,which are the following:

- Calculating u: $u = g^x \ mod \ p$

[3]www.bouncycastle.org

- Calculating the secret: $s = v^x \ mod \ p$
- Encryption time
- Decryption time
- Communication time I
- Communication time II
- Protocol Time

The phone has been resetet after ten measurements, due to a occurring memory shortage.To get results which are at least partly comparable to the theoretical work, the username and password were hard coded into the MIDlet. Username and password are chosen, in a way that they are 128Bit long.
This protocol was run 100 times. And again there were some interesting results.

- Calculation of u: Although 44% of the calculations take between 11 and 20 ms the average calculation takes about 33 ms.
- Calculation of s: At this calculation there is now real peak. The span of time needed is pretty big: The fastest calculation has been done in 13ms while the longest took 2560ms. But the average calculation of the secret takes 240 ms and more than 97% of the calculations are faster than 460ms.
- Encryption time: Differs between 90 and 2328 ms. While the average lies at 190 ms.
- Decryption time: Decryption takes between 63 and 2336 ms, with an average at 147 ms. But it is obvious that encryption and decryption time vary from step to step in about the same way. This might mean, that the times needed is in any kind connected to how the key looks like.
- Communication times: The communication times lie close to each other, with some peaks, that will be explained later. The average communication takes about 2500ms. Within this time included, there is also the time needed for calculations on the Server.
- Protocol Time: The whole authentication takes in average about 7350ms. Whereas the timespan lies between 5400ms and 19720ms. 90% of the authentications finish in less than 9200ms.

Looking at the values shows that every 10th Authentication takes much longer, all single steps require more time:calculations, encryption,decryption and the first communicationstep as well. It takes some time to reinitialize the additional classes. So if the authentications would not run in this kind of batched mode, each one might take about the average time of these ten values, here looking like extreme peaks in this measurement. This would raise the average authentication time to 13750 ms, which is nearly 90% longer.

3.5 Conclusions

After having implemented two of the protocols discussed in the first part. It is now time to draw conclusions.
Comparing the results from the first and the second part it is not that easy to say that everything just could be confirmed. Especially looking at the times needed for the calculations it seems not to work out at all.
But still you could say, that an authentication that is regarded secure than the authentication using the SSH-like implementation, meaning at least 30 Challenge / Responds. The SSH version is still faster even if it is compared to the FS-Short protocol. If you subtract the additional time that comes with the usage of the BigInteger class, it gets even faster. Maybe it would be a good idea to upgrade the Fiat-Shamir implementations, so that they also make use of BigInteger numbers, instead of longs. But then the time needed to authenticate using Fiat-Shamir will probably raise and the difference will become even bigger.

4. Resume and forecast

There is not real result in this work. But you can say that based on the given prerequisites and the needs one of the Algorithms can be chosen.
If the bandwidth is small an algorithm with the minimal data transferred should be suitable. If the small device has very limited memory resources an algortihm which has little values to save is better than for example the FS-Short, which has to deal with a big number of temporary values.
If the small device is only a short period of time within the reach of the proxy, the algorithm needs to be quick. As for example the SSH-like protocol.

So after beeing successfully authenticated this could not be the end. There should be any form of authorization and accounting, as well as any form of secured transmission afterwards.
For example could a device, after it has been authenticated secure its' transmissions by using MAC.
Another possibility might be, that a device first tries to communicate using MAC or similar techniques. If it has not been authenticated previously the Security Proxy Denies all communication and requests a proper authentication first. This Scheme could speed up the communication if the small devices are roaming. A device has only to be authenticated once. But this requires on the other hand much more memory and processor power, to store keys, and calculate MACs for every message.
But these things have to be considered separately.

A. Appendix - First Part

A.1 RSA Public-Key-Algorithm

The RSA Algorithm is the typical example for a Public-Key Algorithm. It uses a system of asymmetric keys. Which means, there exists a key-pair. With a public part and a private part. Messages encrypted whith the public key, can only be decrypted with the corresponding private key.

A.1.1 Creating the keys

This section describes, how the keys are created:

- Choose two very large primes p and q
- Compute $n = p \cdot q$
- Compute $z = (p-1)(q-1)$
- Choose a private key d, being coprime to z
- Compute the corresponding public key e, satisfying $e \cdot\ d = 1 mod\ z$

Now the key-pair is ready for usage.

A.1.2 Encryption and Decryption

After creating the keys, and distributing the public part, messages can be encrypted by anyone and decrypted by the holder of the secret key.

- Messages are Strings of Bits

- Messages are splitted into fixed-length blocks $[m_0 \dots m_i]$
- The bitstrings of the blocks, are interpreted as a number $0 \leq m_i \leq n$
- These blocks are processed to the encrypted message C: $C = [c_0 \dots c_i]$
- While s_i: $c_i = m_i^e \ mod \ n$

- Receiver splits the encrypted message C again into blocks
- Receiver decrypts the single blocks: $m_i = c_i^d mod\ n$
- Receiver combines the blocks to the original message M: $M = [m_0 \dots m_i]$

Using a system based on public keys includes the need of authentic keys. Otherwise it might be possible to encrypt a message to a entity B with a key that in reality belongs to C. Now B gets the message, but can't decrypt it. C has now the possibility to read the message which was send to B and thougt to be confidetal. Because of this fact public-key systems are often connected to certificates. These certificates are released from Trust Centers, which make sure that the key belongs to the entity it is said to belong to.

A.2 DES / 3DES - Encryption

DES is a typical example for a Blockcipher-Algorithm. Unlike RSA uses DES symmetric keys, which means, the same key to encrypt and decrypt the messages. Here it is possible that anyone, owning the key can decrypt and read intercepted messages. This means, that you have to share one key with any communicationpartner.

Since it is known that DES is to be considered unsecure there are several other algorithms. One possibility is to use DES three times on the same message, this method is known as 3DES. But even this method seems not to be used for too long time, due to security problems. There are already newer and possibly better algorithms, like e.g. AES. In this Report DES will be used because it is not to complex.

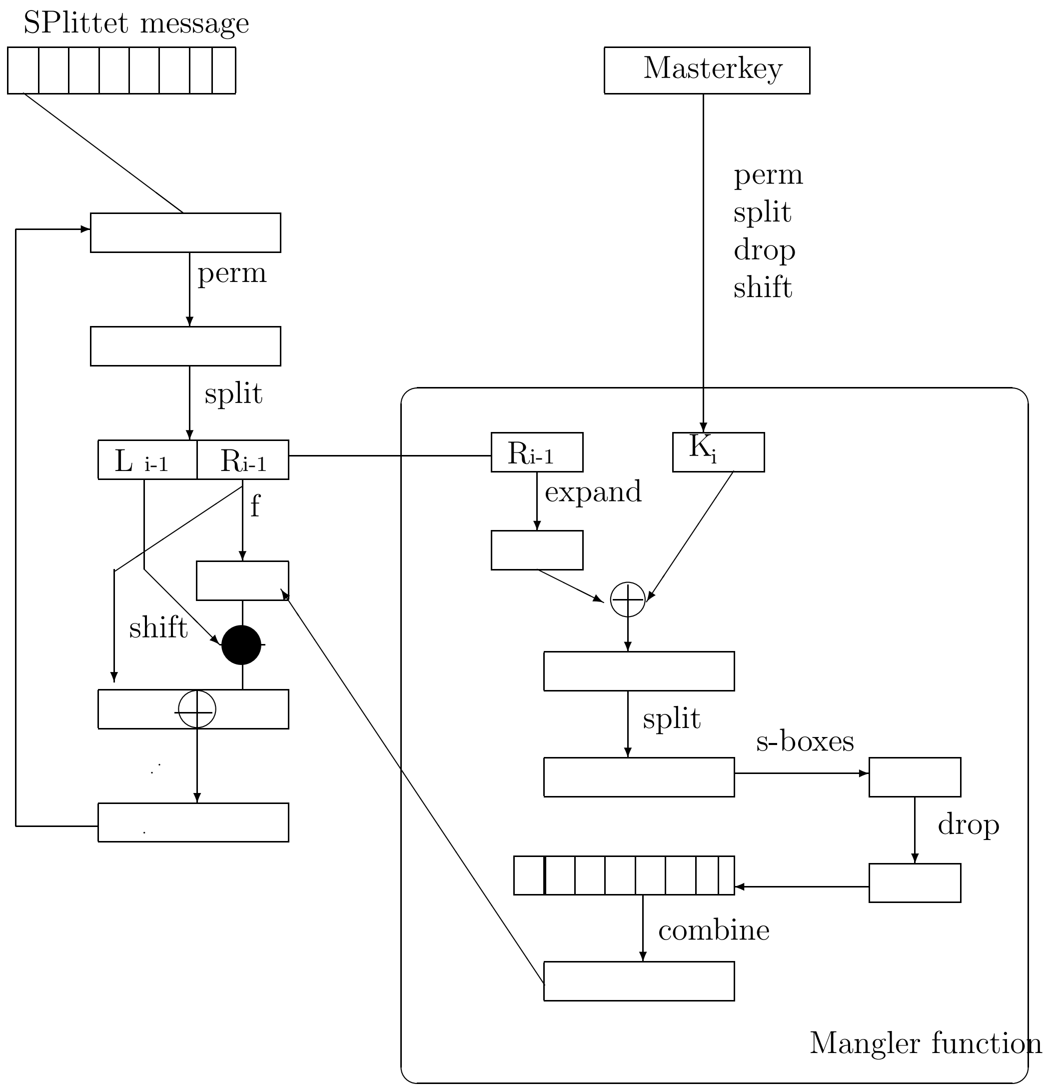
A.2.1 Encryption

- Operates on 64-bit blocks
- Each Block is computed to a 64-bit output-block, within 16 rounds.
- DES uses 48-bit keys
- Each Round uses its' own key
- Each key is derived from a 56-bit Masterkey, which is the shared secret
  - First the key is initially permutated
  - It is divided into two 28-bit blocks
  - For each round the halves are rotated one or two bits to the left
  - 24 bit are extracted and combined to a 48-bit key

The encryption is quite simple:

- 64-bit block is initially permutated
- Round i takes the result of round i-1
- Divides $result_{i-1}$ into two 32-bit parts: L_{i-1}, R_{i-1}
- Computes $L_i = R_{i-1}, R_i = L_{i-1} \bigoplus f(R_{i-1}, K_i$
- $f(R_{i-1}, K_i)$ is called mangler function
  - f() takes 32-bit R_{i-1} and 48-bit key
  - Expands R_{i-1} to 48-bit block
  - XORS it with K_i
  - Cuts result into eight 6-bit chunks
  - $6 - bitinput \rightarrow SBox \rightarrow$ 4-bit output
  - Combines them again into a 32-bit string
  - Final Permutation

To sum up how many cycles are executed it is necessary to have a closer look to the number of basic operations that have to be executed. Down at this level there is again a possibility to compare the algorithm to other algorithms.



SPlittet message
Masterkey
perm
split
drop
shift
perm
split
L i-1
Ri-1
f
shift
Ri-1
Ki
expand
split
s-boxes
drop
combine
Mangler function

To be able to sum the basic operations up, the following assumptions are made:

Operation	Number of Cycles
AND	2
ADD	2
MULT	15
OR/XOR	2
SHIFTX	3
split	6
rotateX	6
expand/combine	0
SBox	s
drop	s
Permutation (perm)	t*6

The number of the SBoxes is expressed through the variable s.

Now the sum for one round is (748 + 10s) cycles, while the sBoxes are not yet analysed. For the whole encryption this value has to be multiplicated with B: $B = \frac{messagelength}{64Bit}$

A.3 MD5 - Hashing

MD5 is cryptographic algorithm to create a 128-bit fixedlength message digest, out of messages with arbitrary length. MD5 is a one way function. It is not possible to recompute the message out of the message digest.

- Message is padded to 448 bit (Modulo 512)
- The length of the message is added, as a 64-bit Integer.
- Starting with some initial 128-bit value
- Algorithm works over k phases (k = length div 512)
- In each phase there is a 128-bit digest computed, using the previous and 512-bit block of the message as inputs
- Each phase consist of 4 rounds, connected to four functions
  - F(x,y,z) = (x AND y) OR ((NOT x) AND z)
  - G(x,y,z) = (x AND z) OR (y (NOT z))
  - H(x,y,z) = x OR y OR z
  - J(x,y,z) = y OR (x or (NOT z))
  - Each of these functions operates on the 32-bit Variables
  - Therefor the 512-bit messageblock is divided into 16 32-bit blocks.

- Each function is used to change four variables (p,q,r,s) in 16 iterations $b_{0...15}$
- Iterations compare Tanenbaum[4][434]

Also this time it is impossible to calculate a number of simple multiplications. So here is again a summation of steps that are to be processed, which can be transformed to basic instructions for a specific platform.

$$\begin{gathered}
K\cdot [16\cdot (p + F(q,r,s) + b_{0...15} + C_{0...15} + rotateX) \\
+16\cdot (p + G(q,r,s) + b_{0...15} + C_{0...15} + rotateX) \\
+16\cdot (p + J(q,r,s) + b_{0...15} + C_{0...15} + rotateX) \\
+16\cdot (p + H(q,r,s) + b_{0...15} + C_{0...15} + rotateX)]
\end{gathered}$$

Using the values assumed in A.2 the summed up cylces are:

$$\begin{gathered}
K\cdot [16\cdot (2+4+2+2+6) \\
+16\cdot (2+4+2+2+6) \\
+16\cdot (2+2+2+2+6) \\
+16\cdot (2+3+2+2+6)] \\
= \underline{K \cdot 976}\ Cycles
\end{gathered}$$

B. Implementation - Results Appendix

B.1 Fiat-Shamir Basic Version Results

Fiat-Shamir-Protocol over one round:

Round	Random	X	Y	Challenge	Overall
1	1	2	1	0	27450
1	1	1	0	1	12356
1	1	2	2	0	28738

Fiat-Shamir-Protocol over five rounds:

First run:

Round	Random	X	Y	Challenge	Overall
1	1	2	0	1	
2	39195	1	2	0	
3	48555	1	0	1	
4	60771	1	0	1	
5	73045	1	1	1	76023

Second run:

Round	Random	X	Y	Challenge	Overall
1	1	2	0	1	
2	53953	1	1	0	
3	64113	1	1	0	
4	74554	1	1	0	
5	84831	1	1	1	97045

Third run:

Round	Random	X	Y	Challenge	Overall
1	1	2	1	0	
2	12347	1	2	0	
3	18558	1	0	1	
4	24787	11	1	0	
5	31051	1	1	0	37459

Forth run:

Round	Random	X	Y	Challenge	Overall
1	1	2	1	0	
2	12347	1	2	0	
3	18558	1	0	1	
4	24787	11	1	0	
5	31051	1	1	0	37459

Fifth run:

Round	Random	X	Y	Challenge	Overall
1	1	2	1	0	
2	10353	1	0	1	
3	15843	3	0	1	
4	21249	1	0	1	
5	26929	1	4	1	33223

Fiat-Shamir over ten rounds

First run:

Round	Random	X	Y	Challenge	Overall
1	1	2	11	1	
2	9742	1	0	1	
3	16030	1	0	1	
4	22364	1	38	0	
5	28573	1	1	1	
6	35551	2	0	1	
7	41710	1	0	1	
8	57925	1	1	1	
9	53937	1	0	0	
10	60111	2	0	1	66240

Second run

Round	Random	X	Y	Challenge	Overall
1	1	1	1	1	
2	12275	2	1	1	
3	17797	29	1	1	
4	27350	1	0	0	
5	33716	1	1	1	
6	39122	1	0	1	
7	44590	1	0	1	
8	50144	2	0	1	
9	55633	1	1	0	
10	62039	1	0	1	65090

Third run:

Round	Random	X	Y	Challenge	Overall
1	1	2	0	1	
2	8628	1	0	1	
3	14300	1	1	0	
4	20111	2	1	1	
5	25957	44	1	0	
6	32102	1	0	1	
7	37753	32	1	0	
8	43515	1	1	1	
9	48978	1	2	0	
10	55035	2	1	0	65090

B.2 Fiat-Shamir Short Version Results

More or less the same measurements are done as at the usual protocols, but the calculation of the random values and the X-values is combined.

FS-Short over three rounds, one message

First run:

Round	Random / X	Y	Challenge	Overall
1	5	2	0	
2	21420	2	1	
3	27399	2	1	30136

Second run:

Round	Random / X	Y	Challenge	Overall
1	5	3	1	
2	17846	2	0	
3	25737	2	0	17307

Third run:

Round	Random / X	Y	Challenge	Overall
1	5	3	0	
2	4996	2	1	
3	11109	2	0	30136

FS-Short over three rounds, five messages

First run:

Round	Random / X	Y	Challenge	Overall
1	15	15	11010	
2	8827	10	01110	
3	17829	15	01011	28129

Second run:

Round	Random / X	Y	Challenge	Overall
1	14	9	10110	
2	18385	10	11011	
3	28548	9	01001	17307

Third run:

Round	Random / X	Y	Challenge	Overall
1	17	10	10110	
2	14317	10	10000	
3	22926	23	00000	30136

FS-Short over three rounds, ten message

First run:

Round	Random / X	Y	Challenge	Overall
1	29	18	1010100111	
2	20861	23	0000101101	
3	29930	48	1110001111	37924

Second run:

Round	Random / X	Y	Challenge	Overall
1	28	50	1001011110	
2	11187	17	1000011110	
3	21510	19	0001100000	28018

Third run:

Round	Random / X	Y	Challenge	Overall
1	29	24	0110011010	
2	16255	52	0000010000	
3	22830	20	0110100000	30160

FS-Short over three rounds, thirty message

First run:

Round	Random / X	Y	Challenge	Overall
1	104	81	110111110010100001111011100000	
2	23685	96	110000100111011100101111010000	
3	33113	124	010011111011111000100101010110	46014

Second run:

Round	Random / X	Y	Challenge	Overall
1	104	88	0001100111100001101010101101001	
2	26116	63	101001010010110101000001000001	
3	40446	80	101101001000001011011100100100	47859

Third run:

Round	Random / X	Y	Challenge	Overall
1	114	145	010111011110010110000111001110	
2	16088	77	101011100110110010101111011110	
3	25553	199	011010100010001000001110010001	39094

B.3 SSH-Measurements

The following Values are shown in this table:

- Roundnumber: Nr
- Time needed for calculating U: U
- Time needed for calculating the secret: S
- Time needed for encryption: E
- Time needed for decryption: D
- Time needed for first communication: C_1
- Time needed for second communication: C_2
- Time needed for the whole authentication: Rt

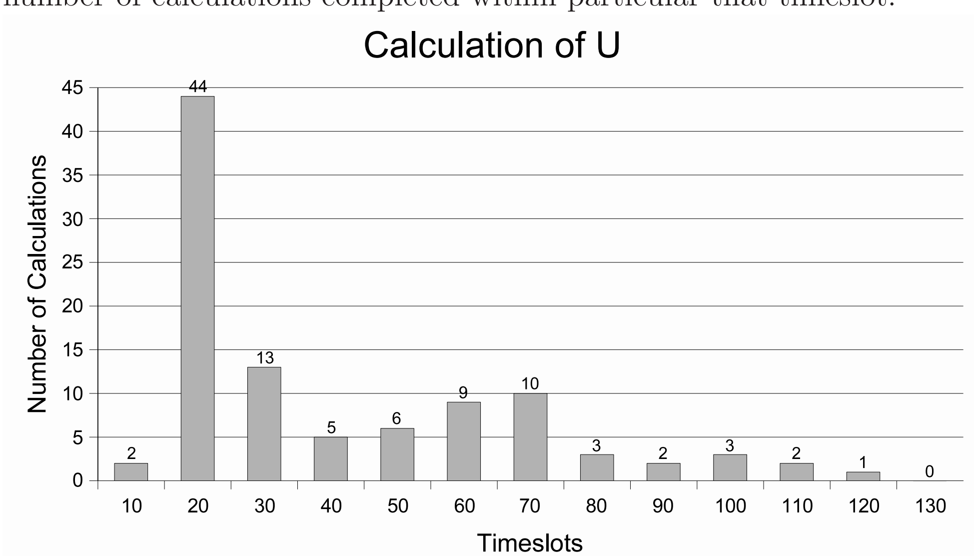
Nr	U	S	E	D	C_1	C_2	Rt
1	105	18	521	72	24302	1957	28623
2	10	175	180	112	2097	2379	6669
3	13	411	171	111	2020	2327	6985
4	49	373	163	104	2399	1984	7016
5	28	507	139	115	1901	2197	6728
6	27	292	90	179	1965	2307	6681
7	14	14	180	198	2404	2456	6618
8	44	260	158	95	2335	4401	8995
9	29	110	131	74	2019	2354	6715
10	20	88	131	113	2166	2146	6549
11	71	30	443	68	6083	1859	9894
12	12	415	135	108	2210	2001	6682
13	13	163	129	102	2796	2055	6828
14	11	91	132	179	2382	2235	6559
15	59	401	161	148	1910	2177	6679
16	12	284	105	178	2578	2270	6944
17	30	400	90	122	2070	2059	6788
18	9	187	166	143	2318	2332	6678
19	50	534	158	71	2057	2323	7207
20	12	49	122	81	1371	4624	7698
21	67	81	525	99	11223	1929	15271
22	76	47	112	115	2075	2486	6806
23	16	206	213	69	1503	4152	7733
24	82	382	186	159	2069	2134	6769
25	11	36	144	128	1984	2452	6476
26	12	28	144	113	2117	2312	6525
27	16	211	131	102	2255	2424	6945
28	52	16	104	163	2040	2425	6660

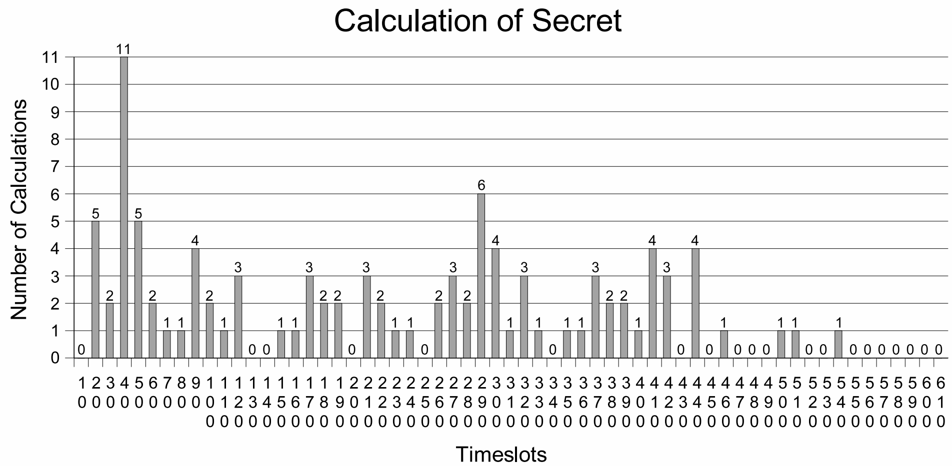
Nr	U	S	E	D	C_1	C_2	Rt
29	11	36	90	179	2079	9507	13814
30	12	406	118	82	2059	1976	6721
31	68	294	389	144	4233	2029	8665
32	66	222	165	134	2572	2312	7042
33	50	113	116	121	2048	2452	6785
34	30	155	136	110	2178	2299	6779
35	23	75	104	163	1961	2355	6578
36	11	208	134	102	1997	2076	6472
37	10	52	157	195	4524	2035	8371
38	16	297	103	117	2333	2569	7349
39	51	48	103	179	1910	2334	6398
40	27	390	108	109	2198	2041	6764
41	104	353	442	106	5585	2040	10244
42	12	290	134	192	2867	2376	7110
43	25	44	2328	95	2306	1991	8619
44	12	275	178	102	2367	2154	6628
45	52	309	108	139	2147	2125	6561
46	11	84	142	107	2341	2410	6907
47	15	434	134	82	1955	2020	9398
48	11	289	156	102	2295	2242	6615
49	13	371	128	104	1967	2157	6655
50	23	34	211	147	2370	2393	6649
51	92	13	481	70	5657	1844	9380
52	15	33	157	103	2376	2329	6491
53	12	318	129	104	2096	2144	6686
54	26	319	132	113	2112	2065	6703
55	14	365	166	185	2139	1988	6655
56	46	100	102	180	2432	2325	6447
57	73	343	100	87	2121	2250	6649
58	68	165	137	102	2295	2431	7009
59	60	401	137	146	2171	1998	6742
60	42	142	151	104	1575	2150	5467
61	66	281	425	191	5168	2254	9687
62	22	280	128	78	2127	2381	6743
63	11	39	175	111	2308	2136	6822
64	64	452	173	176	2029	2108	6795
65	27	33	132	103	2112	2478	6720
66	10	262	130	101	2092	2151	6693
67	13	437	167	178	2020	2129	6615
68	36	172	107	177	2046	2320	6585
69	50	19	178	120	1920	2371	6444
70	27	259	162	103	2081	2208	6635
71	94	161	486	63	5282	2160	9189
72	11	80	177	142	2485	2264	6582
73	53	382	156	103	1929	2249	6745
74	10	238	102	171	2343	2374	6779

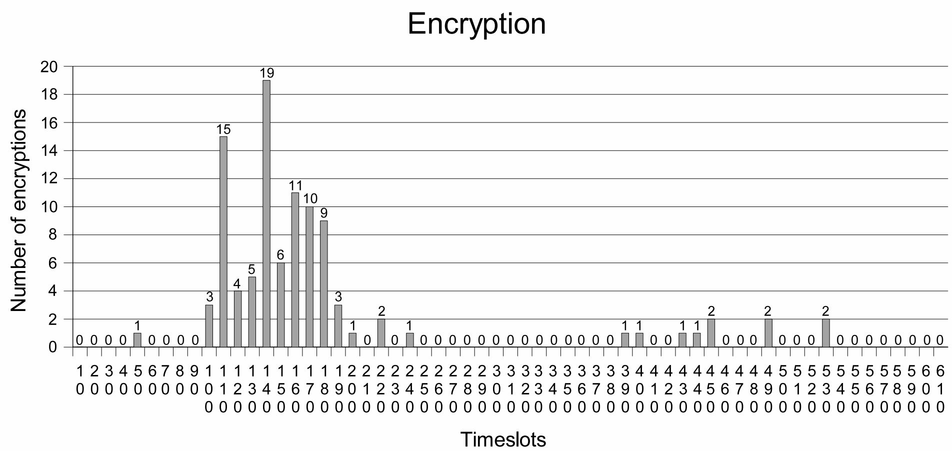
Nr	U	S	E	D	C_1	C_2	Rt
75	28	259	150	84	2040	2112	6654
76	65	43	152	167	1919	2317	6433
77	40	317	173	124	2531	2205	6860
78	11	361	175	102	2546	2083	7102
79	11	278	104	182	2010	2199	6611
80	31	184	108	115	2388	2098	6501
81	97	329	392	112	15055	2202	19711
82	14	57	145	109	1960	2158	6503
83	49	31	135	135	1886	2211	6348
84	14	363	107	106	1916	2298	6823
85	13	65	239	115	2051	2199	6588
86	8	22	110	179	2293	2389	6415
87	13	267	146	135	2250	2196	6767
88	86	436	147	140	2121	1958	6779
89	13	439	135	2336	1869	2251	8893
90	11	217	106	140	2230	2210	6753
91	115	493	433	142	11727	2100	16829
92	12	206	177	133	2098	2214	6604
93	13	118	196	83	2234	2175	6679
94	65	415	166	104	1925	2378	6934
95	62	93	157	83	3312	1927	7481
96	39	2528	159	139	2125	1948	8759
97	12	284	160	188	2059	2214	6704
98	11	280	161	118	2001	2383	7059
99	51	31	136	177	2041	2415	6685
100	10	38	136	178	2380	2395	6461

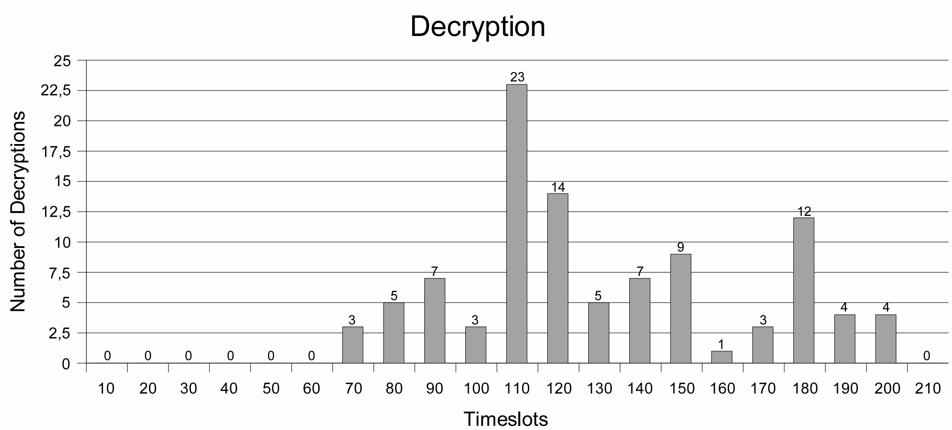
B.3.1 Analysis of the data

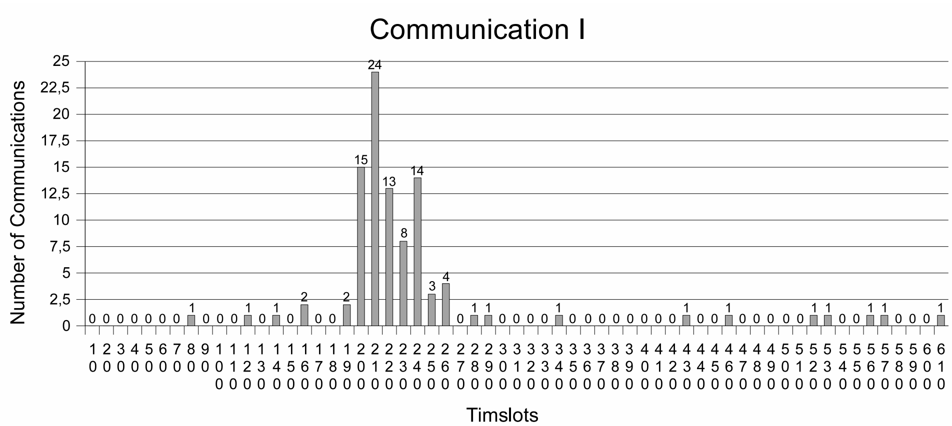
The following diagrams show the distribution of the results within certain timeslots. Each column represents a timeslot of 10ms. Starting at 0ms. The Y-Axis shows the number of calculations completed within particular that timeslot.

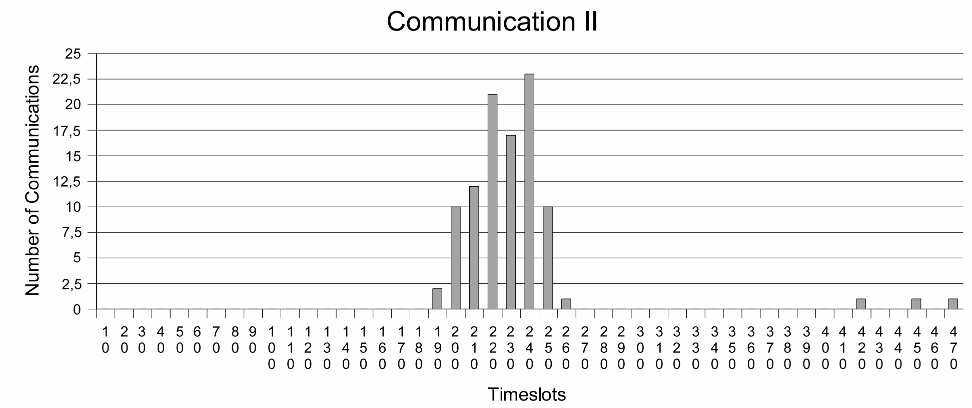


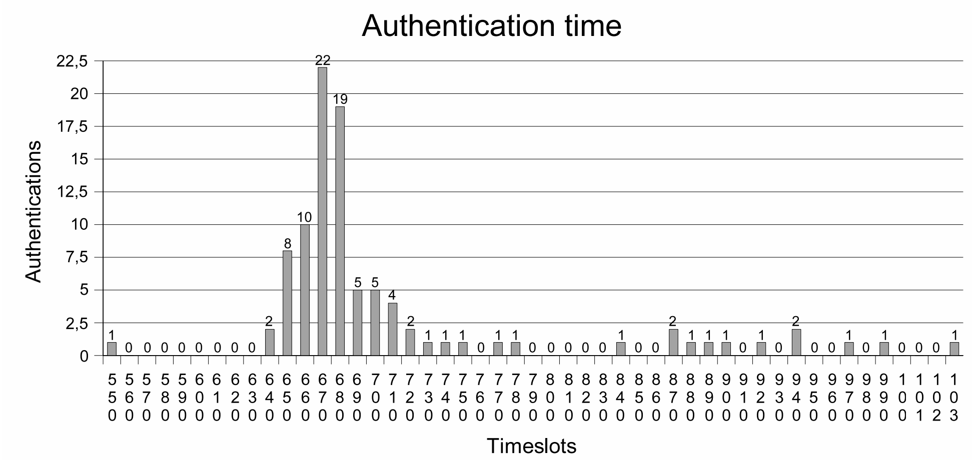

Calculation of Secret
Number of Calculations
Timeslots


Encryption
Number of encryptions
Timeslots


Decryption
Number of Decryptions
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210
Timeslots


Communication I
Number of Communications
Timslots


Communication II
Number of Communications
Timeslots


Authentication time
Authentications
Timeslots

C. Source Code

Here are the most important parts of the source code. For each implementation these parts of the sourcecode are included, that do most of the work!

C.1 Fiat-Shamir

C.1.1 FS-MIDlet

```
package midlets.fiatshamir;

import javax.microedition.lcdui.*;
import java.io.IOException;
import java.util.*;



class FiatShamirScreen
    extends Form
    implements CommandListener, HttpPosterListener
{
    private final FiatShamirMIDlet midlet;
    private final HttpPoster httpPoster;
    private final StringItem outputField;
    private final StringItem outputField1;
    private final StringItem outputField2;
    private final Command quitCommand;
    private volatile boolean readyForInput = true;
    private volatile boolean readyForRead = false;
    public long reply;
    private long r;
    private long rreply;

    FiatShamirScreen(FiatShamirMIDlet midlet,
              HttpPoster httpPoster)
    {
        super("fiatshamir");
        this.midlet = midlet;
        this.httpPoster = httpPoster;
        outputField = new StringItem("Fiat-Shamir Authentificaton!","");

        append(outputField);
        outputField1 = new StringItem("","");
        append(outputField1);

        outputField2 = new StringItem("Times: ","");
        append(outputField2);

        String times = new String();



       //Precomputed and Hardcoded TrustCenter Information
       // p = 41  p=6781
       // q = 13  q=313721

       Long n = new Long(2127342101); // n = p*q
       long s = 76924391; // A's secret coprime to n
       Long v = new Long ((s*s) % n.longValue()); //public key to be registered at the TC
```

```
        //v=1597229406


        //set first timer
        Date start = new Date();
//end first timer

       while (true){

             if ((reply ==11) || (reply==10)){
              break;
              }

            Random rand=new Random();  // initalising the random numebr generator

            r = rand.nextLong() %  n.longValue(); //making sure the random is less than n

            if (r <= 0) {r = r * -1;} // making the random commitment positive

            Date randomT = new Date();
            times = times +("random:"+(randomT.getTime()-start.getTime()));

             Long x = new Long((r*r)%n.longValue()); // x=r^2 mod n

            Date xT = new Date();
            times = times +("  xT:"+(xT.getTime()-randomT.getTime()));

        //send first phase information public key v and commitment x
        try{
               String requestStr = "v="+v.longValue()+"&x="+x.longValue()+"&n="+n.longValue()+"&r="+r;
               httpPoster.sendRequest(requestStr, this);
               readyForInput = false;
               readyForRead = false;
            }
            catch (IOException e)
                {
                  outputField.setText("Error");
                }
        //end first phase
        outputField.setText(""+reply);
        //working on challenge



        while (readyForRead == false)
            {
                //hibernate
            }

            if ((reply ==11) || (reply==10)){
             break;
             }

        long y = 0;  //initalising prove

Date yT1 = new Date();

        if (reply == 0){
                y = r;
        }


        if (reply ==1){
                y = (r * s) % n.longValue();
}

         else{

                 outputField1.setText("1/"+reply);
                 y = r;
             }



Date yT2 = new Date();
        times = times +("   yT:"+(yT2.getTime()-yT1.getTime()));

        //send second phase

        try{
             String requestStr = "y="+y;
             httpPoster.sendRequest(requestStr, this);
             readyForInput = false;
             readyForRead = false;
            }
            catch (IOException e){
                          outputField.setText("Error");
                }


        while (readyForRead == false)
            {
                //hibernate
            }

    }

        if (reply == 10){

                outputField1.setText("Failed");
```

```
}

        if (reply == 11){

                Date stop = new Date();
                outputField1.setText("Succeded");
                outputField.setText(""+(stop.getTime()-start.getTime()));
                outputField2.setText(times);
         }

//end working on challenge

     quitCommand = new Command("Quit", Command.EXIT, 2);
     addCommand(quitCommand);
     setCommandListener(this);
}


public void commandAction(Command c, Displayable d)
    {
        if (c == quitCommand){

            midlet.FiatShamirScreenQuit();
         }
        }


public void receiveHttpResponse(String response){

        reply = Integer.parseInt(response);
        response = "";
        outputField.setText(""+reply);
        readyForInput = true;
        readyForRead = true;

    }


public void handleHttpError(String errorStr){
        outputField.setText("Error");
        readyForInput = true;
    }
}
```

C.1.2 FS-Servlet

```
package servlets.fiatshamir;

/**
 * <p>Überschrift: Servlets Studienarbeit</p>
 * <p>Beschreibung: Servlets zur Studienarbeit</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Organisation: Uppsala Universitetet / Universität Tübingen</p>
 * @author Tobias Bandh
 * @version 1.0
 */

import java.util.Enumeration;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.math.*;


public class FiatShamirServlet
    extends HttpServlet {

//Initialisation
  public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
    }
//end


public void doPost(HttpServletRequest  request,
                      HttpServletResponse response)
       throws IOException, ServletException{

//logfile
File[] rootlist = File.listRoots();
    String path = rootlist[0]+"temp"+File.separatorChar+"data"+File.separatorChar;
     String filename = "fiatshamirlog1.txt"; //datafile
PrintWriter file_out = new PrintWriter(new FileWriter(path+filename,true));
//end logfile

       int phase=0;
       int round=-1;
       int requiredRounds = 10;
       Long n = new Long(2127342101);
       boolean succ = true;
       Integer phaseObject = new Integer(1);
       Integer roundObject = new Integer(1);
       try{

         // First handle session
```

```java
         HttpSession session = request.getSession(false);
         if (session == null) { // first connection?

            phase = 0;
            round = 0;
            session = request.getSession(true); // create
            String requestUrl = HttpUtils.getRequestURL(request).toString();
            String rewrittenUrl = response.encodeURL(requestUrl);
            response.setHeader("X-RewrittenURL", rewrittenUrl);
            session.setAttribute("round", new Integer(round + 1));
            session.setAttribute("phase", new Integer(0));
          }
          else { // if this is not the first connection there must information saved

            phaseObject = ( (Integer) (session.getAttribute("phase")));
            phase = phaseObject.intValue();
            roundObject = ( (Integer) (session.getAttribute("round")));
            round = roundObject.intValue();
}

// read request
InputStream in = request.getInputStream();
         int requestLength = request.getContentLength();

         if (requestLength == -1) {
           throw new IOException("Need to know request length");
         }
         StringBuffer buf = new StringBuffer(requestLength);

         for (int i = 0; i < requestLength; ++i) {
           int ch = in.read();
           if (ch == -1) {
             break;
           }

           buf.append( (char) ch);
         }
         in.close();

         //requestStr contains the parameters and values e.g. v=3&x=23
         String requestStr = buf.toString();

         // process request, producing response
         String responseStr = "";
         //while (round <= requiredRounds) {
         if (round <= requiredRounds) {

           if (phase == 0) { //if it is the first phase
             try {

               StringTokenizer st = new StringTokenizer(requestStr, "&");
               StringTokenizer st1 = new StringTokenizer(st.nextToken(), "=");
               StringTokenizer st2 = new StringTokenizer(st.nextToken(), "=");
               st1.nextToken();
               st2.nextToken();
               long v = Long.parseLong(st1.nextToken()); // get v
               long x = Long.parseLong(st2.nextToken()); // get x
               session.setAttribute("phase", new Integer(1)); //save phase
               session.setAttribute("v", new Long(v)); //save v
               session.setAttribute("x", new Long(x)); //save x

               Random chRand = new Random(); //initialize Random number generator for challenge
               int challenge = chRand.nextInt(2); //create challenge
               responseStr = Integer.toString(challenge); // create response including challenge
               session.setAttribute("e", new Integer(challenge));

               //writing log
               file_out.println("Received x: "+x);
               file_out.println("Send response: " +responseStr);
               file_out.close();
               //end writing log


             }

             catch (NumberFormatException e) {
               responseStr = "Error";
             }
           }

           //else {phase =1;}
           if (phase ==1) { //second phase! receiving prove / checking it now

             StringTokenizer st3 = new StringTokenizer(requestStr, "&");
             StringTokenizer st4 = new StringTokenizer(st3.nextToken(), "=");
             st4.nextToken();
             long y = Long.parseLong(st4.nextToken()); // get y
             Long vv = (Long) (session.getAttribute("v"));
             Long xx = (Long) (session.getAttribute("x"));
             Integer ee = (Integer) (session.getAttribute("e"));
             session.setAttribute("round", new Integer(round + 1));
             session.setAttribute("phase", new Integer(0));
             int e = ee.intValue();
             boolean acc = false;

             //working on the input
             if (y == 0) { //must be rejceted
               acc = false;
             }
             else { //otherwise check input
```

```
            file_out.println("Received y: "+y);
            long ysquare = (y * y) % n.longValue();
            file_out.println("Y^2 mod n: "+ysquare);
            long test = (xx.longValue() *
                        (long) (Math.pow( (double) vv.longValue(), (double) e))) %
                n.longValue();
            file_out.println("Test: "+test);

            if (ysquare == test) {
              responseStr = "" + 1;
              if (round == requiredRounds){
                responseStr = "" + 11;}

              file_out.println("Sent response: " +responseStr);
              file_out.close();

            }
            else
              responseStr = "" + 10;
            file_out.println("Sent response: " +responseStr);
            file_out.close();


          }
          //input verified?

        }
        // write response
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.write(responseStr);
        out.close();



      }
      else {
        // end first connection

        responseStr = "11";

        // write response
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.write(responseStr);
        out.close();
  file_out.close();
      }
    }
      catch (IOException e) {
        e.printStackTrace();
        throw e;
      }
      catch (Exception e) {
        e.printStackTrace();
        throw new ServletException(e.getMessage());
      }

  }



  public String getServletInfo()
    {
      return "SSH Servlet.";
    }
}
```

C.2 Fiat-Shamir Short

C.2.1 FS-Short-MIDlet

```
package midlets.FS_SHORT;

import javax.microedition.lcdui.*;
import java.io.IOException;
import java.util.*;

class FS_SHORT_SCREEN
    extends Form
    implements CommandListener, HttpPosterListener{

    private final FS_SHORT_MIDlet midlet;
    private final HttpPoster httpPoster;
    private final StringItem outputField;
    private final StringItem outputField1;
    private final StringItem outputField2;
    private final StringItem outputFieldX;
    private final Command quitCommand;
    private final Command startCommand;
    private volatile boolean readyForInput = true;
    private volatile boolean readyForRead = false;
    public String reply;
    private long r;
```

```
    private long rreply;
    public String responseStr ="";


    FS_SHORT_SCREEN(FS_SHORT_MIDlet midlet,
              HttpPoster httpPoster){

        super("FS_SHORT");
        this.midlet = midlet;
        this.httpPoster = httpPoster;
outputField = new StringItem("Fiat-Shamir Authentificaton!","");

append(outputField);
outputField1 = new StringItem("","");
append(outputField1);
String requestStr;

outputFieldX = new StringItem("Times: ","");
append(outputFieldX);

String times = new String();

int num_of_com = 30;
reply="";

       Long n = new Long(2127342101); // n = p*q
       long s = 76924391; // A's secret coprime to n
       Long v = new Long (2); //public key to be registered at the TC

       //set first timer
Date start = new Date();
//end first timer


while (true){
            if ((1==(reply.compareTo("11"))) || (1==(reply.compareTo("10")))){
              break;
               }

requestStr="v="+v.longValue();

String r_string ="&r="; //tempstring

long[] rs= new long[num_of_com];
int j = 0;
Random rand=new Random();  // initalising the random numebr generator
while (j <= (num_of_com - 1)){

r = rand.nextLong() % (long) n.longValue(); //making sure the random is less than n
if (r <= 0) {r = r * -1;} // making the random commitment positive
rs[j] = r;

r_string = r_string + Long.toString(r)+"&r=";
   Long x = new Long((r*r)% n.longValue()); // x=r^2 mod n
//building request string
requestStr=requestStr + "&x"+j+"="+x.longValue();
j++;
}


Date randomT = new Date();
times = times +(" random/xes:"+(randomT.getTime()-start.getTime()));

//send first phase information public key v and cimmitment x
try{
r_string = r_string +"0";
requestStr = requestStr;//+ r_string;
httpPoster.sendRequest(requestStr, this);
readyForInput = false;
readyForRead = false;
}
catch (IOException e){
      outputField.setText("Error");
                }

        //end first phase
outputField.setText(""+reply);
//working on challenge

while (readyForRead == false){
//hibernate
}

if ((0==(reply.compareTo("11"))) || (0==(reply.compareTo("10")))){
break;
                 }

String cutStr = "2";
char ch = cutStr.charAt(0);
Tokenize st = new Tokenize(responseStr, ch);

long[] challenges = new long[num_of_com];
requestStr="";
int ch_count = 0;

Date yT = new Date();

while (st.moreT()){
```

```
long y = 0;  //initalising prove
long challenge =Long.parseLong(st.nextT());

if ( challenge == 0){

if (requestStr==""){
  requestStr = "y"+ch_count+"="+ rs[ch_count];
  }
else{
requestStr =requestStr + "&y"+ch_count+"="+ rs[ch_count];
}

}

if (challenge ==1){

y=0;
y = (rs[ch_count] * s) % n.longValue();

if (requestStr==""){
requestStr = "y"+ch_count+"="+ y;}
else{
requestStr =requestStr  +"&y"+ch_count+"="+ y;
}
}

ch_count++;
}


Date yT1 = new Date();
times = times +("  yT:"+(yT1.getTime()-yT.getTime()));

//send second phase

try{
   httpPoster.sendRequest(requestStr, this);
   readyForInput = false;
   readyForRead = false;
}

catch (IOException e){
      outputField.setText("Error");
                }


while (readyForRead == false){
//hibernate
}

responseStr = outputField.getText();


}

if (reply.compareTo("10")==0){

outputField1.setText("Failed");
}

if (reply.compareTo("11")==0){

outputField1.setText("Succeded");
Date stop = new Date();
outputField.setText(""+(stop.getTime()-start.getTime()));
outputFieldX.setText(times);
}


outputField2 = new StringItem("Now ready to receive Messages!","");
startCommand = new Command("OK", Command.OK, 2);
quitCommand = new Command("Quit", Command.EXIT, 1);

if (reply.compareTo("11")==0){

append(outputField2);
addCommand(startCommand);
}

addCommand(quitCommand);
        setCommandListener(this);
 }


public void commandAction(Command c, Displayable d){

        if (c == startCommand){

            //midlet.FS_SHORT_SCREENQuit();
            midlet.FS_SHORT_SCREENDone();
         }
        else {
midlet.FS_SHORT_SCREENQuit();
}

   }


public void receiveHttpResponse(String response){
```

```
        //reply = Long.parseLong(response);
reply ="";
reply = reply +response;
        responseStr= response.trim();
        //outputField.setText(response);
        outputField.setText(""+reply);
        readyForInput = true;
        readyForRead = true;

    }


public void handleHttpError(String errorStr){

        outputField.setText("Error");
        readyForInput = true;
    }
}
```

C.2.2 FS-Short-Servlet

```
package servlets.fs_short;

/**
 * <p>Überschrift: Servlets Studienarbeit</p>
 * <p>Beschreibung: Servlets zur Studienarbeit</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Organisation: Uppsala Universitetet / Universität Tübingen</p>
 * @author Tobias Bandh
 * @version 1.0
 */

import java.util.Enumeration;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.math.*;


public class fs_short_servlet
    extends HttpServlet {



public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
    }



public void doPost(HttpServletRequest  request,
                      HttpServletResponse response)
       throws IOException, ServletException{


 //logfile
File[] rootlist = File.listRoots();
String path = rootlist[0]+"temp"+File.separatorChar+"data"+File.separatorChar;
String filename = "FS_SHORT.txt"; //datafile
PrintWriter file_out = new PrintWriter(new FileWriter(path+filename,true));

//end logfile


int phase = 0;
int round = -1;
int requiredRounds = 3;
int ch_count=30;
Vector requestVec;
boolean succ = true;
Integer phaseObject = new Integer(1);
Integer roundObject = new Integer(1);

try {
       // First handle session
       HttpSession session = request.getSession(false);
       if (session == null) { // first connection?

         phase = 0;
         round = 0;
         session = request.getSession(true); // create
         String requestUrl = HttpUtils.getRequestURL(request).toString();
         String rewrittenUrl = response.encodeURL(requestUrl);
         response.setHeader("X-RewrittenURL", rewrittenUrl);
         session.setAttribute("round", new Integer(round));
         session.setAttribute("phase", new Integer(0));
         session.setAttribute("authenticated", new Integer(0));
        }
        // if this is not the first connection there must information saved
        else {

          phaseObject = ( (Integer) (session.getAttribute("phase")));
          phase = phaseObject.intValue();
          roundObject = ( (Integer) (session.getAttribute("round")));
          round = roundObject.intValue();
```

```
}

        // read request
        InputStream in = request.getInputStream();
        int requestLength = request.getContentLength();
        if (requestLength == -1) {
          throw new IOException("Need to know request length");
        }

        StringBuffer buf = new StringBuffer(requestLength);
        for (int i = 0; i < requestLength; ++i) {

        int ch = in.read();
        if (ch == -1) {
            break;
          }

          buf.append( (char) ch);
        }

        in.close();

        //requestStr contains the parameters and values e.g. v=3&x=23
        String requestStr = buf.toString();

        // process request, producing response
        String responseStr = "";
        //while (round <= requiredRounds) {
        if (round < requiredRounds) {


          file_out.println("Round:" +round);

          if (phase == 0) { //if it is the first phase
            try {

       file_out.println("Phase: "+phase);
requestVec = inputs_split(requestStr);

              int v_number = Integer.parseInt(requestVec.elementAt(0).toString()); // get v
              long v = get_public_key(v_number);
              long n = get_n(v_number);
              Vector xes = new Vector();

              int xcount = 1;
              long xn;
              while (xcount <= requestVec.size() - 1) {
                xes.add(new Long(Long.parseLong(requestVec.elementAt(xcount).
                                                toString())));

file_out.println("received x: "+requestVec.elementAt(xcount).
                                              toString());
               xcount++;
                 }

session.setAttribute("phase", new Integer(1)); //save phase
session.setAttribute("v", new Long(v)); //save v
              session.setAttribute("n", new Long(n));
              session.setAttribute("x", xes); //save xes

Random chRand = new Random(); //initialize Random number generator for challenge
Vector e_Vec = new Vector();

//create challengestring with 5 (ch_count challenges and 2 as seperator
for (xcount = 0; xcount <= (ch_count-1); xcount++) {

int challenge = chRand.nextInt(2); //create challenge
                e_Vec.add(new Integer(challenge));
                if (responseStr == "") {
                  responseStr = responseStr + Integer.toString(challenge);

                }
                else {
                  responseStr = responseStr + "2" + Integer.toString(challenge);
                }
                file_out.println("Challenge: "+challenge);
              }

session.setAttribute("e", e_Vec);
            }

            catch (NumberFormatException e) {
              responseStr = "Error";
            }
          }

            if (phase == 1) { //second phase! receiving prove / checking it now

               file_out.println("Phase: "+phase);
Vector y_es = new Vector();

requestVec = inputs_split(requestStr);
         int xcount = 0;
         long xn;

         while (xcount <= requestVec.size() - 1) { //building y vector
         y_es.add(new Long(Long.parseLong(requestVec.elementAt(xcount).
                                              toString())));
         xcount++;
            }
```

```
            Long vv = (Long) (session.getAttribute("v"));
            Long nn = (Long) (session.getAttribute("n"));
            Vector xx = (Vector) (session.getAttribute("x"));
            Vector ee = (Vector) (session.getAttribute("e"));

  long t_v = vv.longValue();
  long t_n =nn.longValue();

  session.setAttribute("round", new Integer(round+1));
         session.setAttribute("phase", new Integer(0));

         xcount = 0;
            boolean acc = false;

            while (xcount <= (ch_count-1)) {
              int t_e = Integer.parseInt(ee.elementAt(xcount).toString()); //get temporary e
              long t_y = Long.parseLong(y_es.elementAt(xcount).toString()); //get temporary y

              if (t_y == 0) { //must be rejceted
                acc = false;
              }
              else { //otherwise check input
               long ysquare = (t_y * t_y) % t_n;
               file_out.println("y: "+t_y);
                  file_out.println("n: "+t_n);

 //getting temporary y
  long t_x = Long.parseLong(xx.elementAt(xcount).toString());

                long test = (t_x * (long) (Math.pow( (double) t_v, (double) t_e))) %t_n;

                file_out.println("Saved x: "+t_x);
                file_out.println("Y^2: "+ysquare);

                file_out.println("Test: "+test);

if ((ysquare == test) && (responseStr != "10")) {
                  responseStr = "" + 1;
                  if (round+1 == requiredRounds){
                    responseStr = "" + 11;
                    session.setAttribute("authenticated", new Integer(1));

                  }

                }
                else {
                  responseStr = "" + 10;
                              }

              }
                xcount++;
             }

             //working on the input
           }

           // write response
           response.setContentType("text/plain");
           PrintWriter out = response.getWriter();
           out.write(responseStr);
           out.close();
           file_out.close();

       }

       //Processing after authentication
        Integer auth = (Integer) (session.getAttribute("authenticated"));
       if ((round>=requiredRounds) && (auth.intValue() ==0 )){
       responseStr = "11";

       // write response
       response.setContentType("text/plain");
       PrintWriter out = response.getWriter();
       out.write(responseStr);
       out.close();
       file_out.close();
       session.setAttribute("authenticated", new Integer(1));
}
       if  ((round>=requiredRounds) && (auth.intValue() ==1 )){
          Long vv = (Long) (session.getAttribute("v"));
          long t_v = vv.longValue();

          PreparePush message = new PreparePush(t_v);
          responseStr = "";
          if (message.MoreMessages()){
           responseStr = message.NextMessage();
         }
         else
         {
           responseStr = "no_more_messages";
         }


         response.setContentType("text/plain");
         PrintWriter out = response.getWriter();
         out.write(responseStr);
         out.close();
```

```
      }
    }
      catch (IOException e) {
        e.printStackTrace();
        throw e;
      }
      catch (Exception e) {
        e.printStackTrace();
        throw new ServletException(e.getMessage());
      }

    }



/*
  ***************************************************************************

  Accepts a request String
 Returns: A Vector of Arrays
 Array: Parameter Name / Parameter Value

 ***************************************************************************
*/

public Vector inputs_split(String requestStr)
   {
     Vector return_vec = new Vector();
     String[] elements = new String[2];
     StringTokenizer st = new StringTokenizer(requestStr, "&");

  while (st.hasMoreTokens()) {

    StringTokenizer st1 = new StringTokenizer(st.nextToken(), "=");
    st1.nextToken();
    return_vec.add(st1.nextToken());
   }


     return return_vec;
  }

/*
      **********************************************************



      ***********************************************************
*/

public long get_public_key(int i){

       long[] keys = new long[]{237,1597229406};
       return keys[i-1];
     }

public long get_n(int i){

       long[] n_s = new long[] {533, 2127342101};
return n_s[i-1];
     }



   public String getServletInfo()
     {
         return "SSH Servlet.";
     }

}
```

C.3 SSH

C.3.1 SSH-MIDlet

```
package midlets.SSH;

import javax.microedition.lcdui.*;
import java.io.IOException;
import java.util.*;
import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.digests.*;
import org.bouncycastle.crypto.engines.*;
import org.bouncycastle.crypto.paddings.*;
import org.bouncycastle.crypto.params.*;
import org.bouncycastle.util.encoders.Hex;


class SSHSCREEN
    extends Form
```

```
    implements CommandListener, HttpPosterListener{
    private final SSHMIDlet midlet;
    private final HttpPoster httpPoster;
    private final TextField userField;
    private final TextField passField;
    private final StringItem outputField;
    private final Command quitCommand;
    private final Command loginCommand;
    private volatile boolean readyForInput = true;
    private volatile boolean readyForRead = false;

    public String reply;
    String requestStr = "";
    BigInteger secret = new BigInteger("0");

    SSHSCREEN(SSHMIDlet midlet, HttpPoster httpPoster){

        super("SSH");
        this.midlet = midlet;
        this.httpPoster = httpPoster;

outputField = new StringItem("Secret Key","");

append(outputField);

//Prime p
BigInteger p = new BigInteger("36413321723440003717");


//small Value q = 2
BigInteger q = new BigInteger("2");

//Secret x <= 63;
Random rand = new Random();
int x = 0;
while (x==0){
x = rand.nextInt()% 180;
}

if (x<=0){
x=x*-1;
}


//Calculate u
BigInteger exponent = new BigInteger(Integer.toString(x));

BigInteger u = q.modPow(exponent,p);
Pow uu = new Pow(q,exponent,p);


String message = "q="+q.toString()+"&p="+p.toString()+"&u="+u.toString()+"&x1="+Integer.toString(x)+"&uu="+uu.getValue()+"&Steps="+uu.getSteps();


//send message to the server
try{
requestStr = message;
httpPoster.sendRequest(requestStr, this);
readyForInput = false;
readyForRead = false;
    }
catch (IOException e){
       //outputField.setText("Error");
                 }


//wait until repsonse received
while (readyForRead == false){
}

//work on the response

BigInteger v = new BigInteger(reply);
secret = v.modPow(exponent,p);

outputField.setText(""+secret);


//Getting user and Password
userField = new TextField("User", null, 16, TextField.ANY);
        append(userField);
        passField = new TextField("Password", null, 8, TextField.PASSWORD);
        append(passField);


quitCommand = new Command("Quit", Command.EXIT, 2);
        addCommand(quitCommand);
        loginCommand = new Command("Login", Command.SCREEN, 2);
        addCommand(loginCommand);

        setCommandListener(this);
}


    public void commandAction(Command c, Displayable d){

if (c == quitCommand)
        {
            midlet.SSHSCREENQuit();
        }
```

```
        else if (readyForInput)
        {
            if (c == loginCommand)
            {
String plainText = "user="+userField.getString()+"&pass="+passField.getString();
String secretText = doEncryption(plainText,outputField.getText(),outputField.getText());
secretText.trim();

try{
requestStr = secretText;
httpPoster.sendRequest(requestStr, this);
readyForInput = false;
readyForRead = false;
}
catch (IOException e){
       //outputField.setText("Error");
}

while (!readyForRead){
}

String encodedText=reply.trim();
String pass = ""+secret.toString();
String decodedMessage = decodeMessage(encodedText.getBytes(),pass,pass);
outputField.setText(decodedMessage.trim());
}
       }
     }


    public void receiveHttpResponse(String response){

        reply = response;
        readyForRead=true;
        readyForInput=true;

}


    public void handleHttpError(String errorStr){

        readyForInput = true;
     }


    private static String doEncryption(String plainText, String password, String nonce){

String compundKey = password + ":" + nonce;
Digest digest = new MD5Digest();
byte[] key = new byte[digest.getDigestSize()];
digest.update(compundKey.getBytes(), 0, compundKey.getBytes().length);
digest.doFinal(key, 0);

byte content[] = plainText.getBytes();

BufferedBlockCipher cipherEngine = new PaddedBufferedBlockCipher(new DESEngine());
cipherEngine.init(true, new KeyParameter(key));

byte[] cipherText = new byte[cipherEngine.getOutputSize(content.length)];

int cipherTextLength = cipherEngine.processBytes(content, 0, content.length,
cipherText, 0);
try{
cipherEngine.doFinal(cipherText, cipherTextLength);
}
catch(InvalidCipherTextException e){
}

return new String(Hex.encode(cipherText));

     }


private static String decodeMessage(byte[] content, String password, String nonce){

String compundKey = password + ":" + nonce;
Digest digest = new MD5Digest();
byte[] key = new byte[digest.getDigestSize()];
digest.update(compundKey.getBytes(), 0, compundKey.getBytes().length);
digest.doFinal(key, 0);

byte cipherText[] = Hex.decode(content);
BufferedBlockCipher cipherEngine = new PaddedBufferedBlockCipher(new DESEngine());
cipherEngine.init(false, new KeyParameter(key));

byte[] plainText = new byte[cipherEngine.getOutputSize(cipherText.length)];

int plainTextLength = cipherEngine.processBytes(cipherText, 0, cipherText.length,
plainText, 0);
try{
cipherEngine.doFinal(plainText, plainTextLength);
      }
  catch(InvalidCipherTextException e){
}

return new String(plainText);

    }
}
```

C.3.2 SSH-Servlet

```
package servlets.ssh;

/**
 * <p>Überschrift: Servlets Studienarbeit</p>
 * <p>Beschreibung: Servlets zur Studienarbeit</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Organisation: Uppsala Universitetet / Universität Tübingen</p>
 * @author Tobias Bandh
 * @version 1.0
 */

import java.util.Enumeration;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.math.*;


import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.digests.*;
import org.bouncycastle.crypto.engines.*;
import org.bouncycastle.crypto.paddings.*;
import org.bouncycastle.crypto.params.*;
import org.bouncycastle.util.encoders.Hex;


public class ssh
    extends HttpServlet {

//Initialisation
  public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
    }


  public void doPost(HttpServletRequest  request,
                      HttpServletResponse response)
              throws IOException, ServletException{



//logfile
            File[] rootlist = File.listRoots();
            String path = rootlist[0]+"temp"+File.separatorChar+"data"+File.separatorChar;
            String filename = "SSH.txt"; //datafile
            PrintWriter file_out = new PrintWriter(new FileWriter(path+filename,true));
//end logfile


//div variables

// phase --> shows in which phase the protocol is
int phase = 0;
Integer phaseObject = new Integer(0);

//secret Key
BigInteger s=new BigInteger("0");
BigInteger secretObject = new BigInteger(s.toString());

     try {
       // First handle session
       HttpSession session = request.getSession(false);
       if (session == null) { // first connection?

         session = request.getSession(true); // create
         String requestUrl = HttpUtils.getRequestURL(request).toString();
         String rewrittenUrl = response.encodeURL(requestUrl);
         response.setHeader("X-RewrittenURL", rewrittenUrl);

         //save phase for next phase
         session.setAttribute("phase", new Integer(1));
         session.setAttribute("round", new Integer(1));
       }

        else { // if this is not the first connection there must information saved

          //get Phase
          phaseObject = ( (Integer) (session.getAttribute("phase")));
          phase = phaseObject.intValue();
          secretObject = ((BigInteger) (session.getAttribute("secret")));
          s = new BigInteger(secretObject.toString());
          Integer roundObject = ((Integer) (session.getAttribute("round")));
          int r = roundObject.intValue();
}


        // read request
        InputStream in = request.getInputStream();
        int requestLength = request.getContentLength();
        if (requestLength == -1) {
          throw new IOException("Need to know request length");
        }

        StringBuffer buf = new StringBuffer(requestLength);
        for (int i = 0; i < requestLength; ++i) {
          int ch = in.read();
```

```
          if (ch == -1) {
            break;
          }
          buf.append( (char) ch);
        }

        in.close();

        //requestStr contains the parameters and values e.g. v=3&x=23
        String requestStr = buf.toString();

        // process request, producing response
        String responseStr = "";

  //working on the phases.

        String values;
        BigInteger  p = new BigInteger("0");
        BigInteger  q = new BigInteger("0");
        BigInteger  u = new BigInteger("0");
        boolean test = false;


switch(phase){

         case 0:
           //expecting q,p,u
          file_out.println("requestStr: "+requestStr);
          values = get_values(requestStr,4);
          file_out.println(values);
          StringTokenizer vt = new StringTokenizer(values,"@");
          q = new BigInteger(vt.nextElement().toString());
          p = new BigInteger(vt.nextElement().toString());
          u = new BigInteger(vt.nextElement().toString());
          long y=0;
          BigInteger res = new BigInteger("0");

          Random rand = new Random();
          BigInteger yy = new BigInteger("0");
          while (test == false){
          while(y<=0){
            y = rand.nextInt(200);}
            file_out.println("Y: "+y);

          if ((y!=0))// &&((t*y)<=160)) {
            yy = new BigInteger(Long.toString(y));
            s= u.modPow(yy,p);
            test=true;
            }
            }

res = q.modPow(yy,p);
           responseStr=""+res;
           session.setAttribute("secret", new BigInteger(s.toString()));
           session.setAttribute("phase", new Integer(1));
           session.setAttribute("u", u);
           session.setAttribute("v", res);
           session.setAttribute("x", new Integer(vt.nextElement().toString()));
           session.setAttribute("y", new Long(y));

          file_out.println("SecretKey: "+s.toString());
           break;

           //second Phase Verifying login and password
           case 1:
             secretObject = ((BigInteger) (session.getAttribute("secret")));
             s = new BigInteger(secretObject.toString());

             //expecting crypted String

               String pass = ""+s.toString();
               requestStr.trim();
               String decodedMessage = decodeMessage(requestStr.getBytes(),pass,pass);

decodedMessage.trim();
             boolean veryf = verifyUser(decodedMessage.trim());
             file_out.println("decodedMessage: "+decodedMessage.trim());
             if(veryf==true){

             String plainText="OK";
             String secretText = doEncryption(plainText,pass,pass);
             responseStr=secretText.trim();
             }

            else {

               String plainText="DENIED";
               String secretText = doEncryption(plainText,pass,pass);
               responseStr=secretText.trim();
             }

session.setAttribute("phase", new Integer(2));

   //responseStr =decodedMessage;
           break;


case 2:
```

```
String logString = "";

String filename1 = "SSH.csv"; //datafile
           PrintWriter csv_out = new PrintWriter(new FileWriter(path+filename1,true));
BigInteger sObject = ((BigInteger) (session.getAttribute("secret")));
             Integer rObject = ((Integer) (session.getAttribute("round")));
             BigInteger uObject = ((BigInteger) (session.getAttribute("u")));
             BigInteger vObject = ((BigInteger) (session.getAttribute("v")));
Integer xObject = ((Integer) (session.getAttribute("x")));
Long yObject = ((Long) (session.getAttribute("y")));

//create String for Logfile round;tU;tS;u;v;s;x;y;
logString = rObject.toString() +";"+ requestStr.trim()+";'" + uObject.toString()+";'" + vObject.toString()
+";'"+sObject.toString()+";"+xObject.toString() +";"+ yObject.toString()+";";


csv_out.println(logString);
csv_out.close();
session.setAttribute("phase", new Integer(0));

int r = rObject.intValue();
session.setAttribute("round", new Integer(r+1));
break;


        }


          response.setContentType("text/plain");
          PrintWriter out = response.getWriter();
          out.write(responseStr);
          out.close();

          file_out.println("ResponseStr: "+responseStr);
          file_out.close();

      }
        catch (IOException e) {
          e.printStackTrace();
          throw e;
        }
        catch (Exception e) {
          e.printStackTrace();
          throw new ServletException(e.getMessage());
            }

}


/*
      **********************************************************

Get Values extracts Values out of a request String

Take request String / Numer of Values

Returns a @ separated list of Values
      ***********************************************************
*/



public String get_values(String requestStr, int count){

        String values ="";
        StringTokenizer st = new StringTokenizer(requestStr, "&");

        int i = 0;

        while(i<=(count-1)){
          StringTokenizer st1 = new StringTokenizer(st.nextToken(),"=");
          st1.nextToken();
          if (i==0){
            values = st1.nextToken().toString();
          }
          else{
    values = values +"@"+ st1.nextToken().toString();
         }
          i++;
        }


       return values;

 }


   public String getServletInfo()
     {
         return "SSH Servlet.";
     }


private static String doEncryption(String plainText, String password, String nonce){

String compundKey = password + ":" + nonce;
Digest digest = new MD5Digest();
byte[] key = new byte[digest.getDigestSize()];
digest.update(compundKey.getBytes(), 0, compundKey.getBytes().length);
digest.doFinal(key, 0);
```

```
byte content[] = plainText.getBytes();

BufferedBlockCipher cipherEngine = new PaddedBufferedBlockCipher(new DESEngine());
cipherEngine.init(true, new KeyParameter(key));

byte[] cipherText = new byte[cipherEngine.getOutputSize(content.length)];

int cipherTextLength = cipherEngine.processBytes(content, 0, content.length,
cipherText, 0);
try{
cipherEngine.doFinal(cipherText, cipherTextLength);
}
catch(InvalidCipherTextException e){
}

return new String(Hex.encode(cipherText));

     }


private static String decodeMessage(byte[] content, String password, String nonce){

String compundKey = password + ":" + nonce;
Digest digest = new MD5Digest();
byte[] key = new byte[digest.getDigestSize()];
digest.update(compundKey.getBytes(), 0, compundKey.getBytes().length);
digest.doFinal(key, 0);

byte cipherText[] = Hex.decode(content);
BufferedBlockCipher cipherEngine = new PaddedBufferedBlockCipher(new DESEngine());
cipherEngine.init(false, new KeyParameter(key));

byte[] plainText = new byte[cipherEngine.getOutputSize(cipherText.length)];

int plainTextLength = cipherEngine.processBytes(cipherText, 0, cipherText.length,
plainText, 0);
try{
cipherEngine.doFinal(plainText, plainTextLength);
      }
  catch(InvalidCipherTextException e){
}

return new String(plainText);

    }


public boolean verifyUser(String message){

      boolean verif= false;
      StringTokenizer st = new StringTokenizer(message,"&");
      StringTokenizer st1 = new StringTokenizer(st.nextToken(),"=");
      st1.nextToken();
      String user = st1.nextToken().toString();
      StringTokenizer st2 = new StringTokenizer(st.nextToken(),"=");
      st2.nextToken();
      String pass= st2.nextToken().toString();
      verif = check_user_db(user,pass);

      return verif;
    }


public boolean check_user_db(String user, String pass){

      boolean verif = false;
      Vector user_db = new Vector();
      String[] users = new String[2];
      users[0] = "442c1963bd105698";
      users[1] = "9a2990e652cc9580";

      user_db.addElement(new String(users[0]));
      user_db.addElement(new String(users[1]));
      int i =0;

      while(i<=user_db.size()-1){

       if (user.compareTo(user_db.elementAt(i))==0){
           i++;
         if(pass.compareTo(user_db.elementAt(i))==0){
           verif=true;
         }

        }
        else{
         i=i+2;
         }
      }

      return verif;
    }


 }
```

Bibliography

[1] *Putty SSH-Client Source.* Http://www.chiark.greenend.org.uk/ sgtatham/putty/.

[2] Fiat A., Shamir A. *How to prove yourself:Practical Solutions to identification and signature problems.*

[3] P. van Oorschot A, Menezes, S. Vanstone. *Handbook of Applied Cryptography.* CRC press, 1996. Www.cacr.math.uwaterloo.ca/hac.

[4] M. van Steen A. S. Tanenbaum. *Distributed Systems - Principles and Paradigmas.* Prentice Hall, 2002.

[5] W. Diffie, M. Hellman. *New Directions in Cryptography.* November 1976.

[6] A. Otto E. Salomonsson O. Wibling O. Widell K. Johnsson, K. Madsen. *Security Proxy - Project Report.* Datakom2 Course, Uppsala University, june 2003.

[7] Nokia. *Getting Started with Java V1.1.* Www.forum.nokia.com.

[8] W. Stallings. *Cryptography and Network Security.* Prentice Hall, 2003. Third International Edition.