NTNU – Trondheim
Norwegian University of
Science and Technology

Password-based digital signatures

Sangeepan
Sivagnanasuntharam

Master of Science in Communication Technology
Submission date: June 2013
Supervisor: Danilo Gligoroski, ITEM

Norwegian University of Science and Technology
Department of Telematics

NTNU – Trondheim
Norwegian University of
Science and Technology

Improving the security of MinID-like login systems

Sangeepan Sivagnanasuntharam

Submission date:	June 2013
Responsible professor:	Danilo Gligoroski, ITEM
Supervisor:	Kristian Gjøsteen, IMF

Norwegian University of Science and Technology
Department of Telematics

Title: Improving the security of MinID-like login systems
Student: Sangeepan Sivagnanasuntharam

Problem description:

Professor Kristian Gjøsteen from the Department of Mathematics proposed a new protocol between a server and a client where the signing key is secret-shared among the two such that the client's share is a specified password. This protocol can be used to improve the existing login systems like MinID.

The following work in done in this thesis:
1. Design the architecture for the system.
2. Write server code for running the server part of the protocol.
3. Write JavaScript code for the client.

During this thesis the student needs to have programming skills (JavaScript and PHP), ability to understand and use mathematical theorems, and perform original research.

Assignment given: January 16, 2013
Responsible professor: Danilo Gligoroski, ITEM
Supervisor: Kristian Gjøsteen, IMF

Abstract

This thesis is a continuation of the project thesis. A substantial part of the work from last semester had to be scrapped due to security issues, but the code for `iframe` has been carried with.

During this thesis I implemented a password-based digital signature scheme proposed by Kristian Gjøsteen and Øystein Thuen [6]. The paper describes the scheme with RSA as the encryption. But since RSA is computationally intensive, it does not fit well on mobile devices running on battery and limited computing capacity. I will instead use elliptic curve cryptography (ECC).
JavaScript is the chosen programming language for the client side since Java is not supported by the most mobile operating systems. The server-side code will be written in PHP. This thesis describes the process of developing fully functioning client- and server-side prototypes. It explains the architectural and technological decisions made before and during the course of the thesis.

The outcome of the thesis is a robust login system which with additional tweaking, is ready to deployed as authentication in stores, e-voting systems and online banking systems.

(0.1)

Sammendrag

Denne oppgaven er en fortsettelse av prosjektoppgaven fra forrige semester. En vesentlig del av arbeidet kunne ikke brukes ettersom den hadde en kritisk sikkerhetsmessig feil, men koden for IFRAME har blitt gjenbrukt her.

I løpet av denne oppgaven har jeg implementert et passord-basert elektronisk signatursystem beskrevet av Kristian Gjøsteen og Øystein Thuen. I artikkelen [] foreslår de et signatursystem med RSA som krypteringsalgoritmen. Men siden RSA krever relativ stor prosessorkraft for å generere private nøkler, passer dette dårlig for det voksende markedet av håndholdte enheter som går på batteri og har begrenset med prosessorkraft. Istedet bruker jeg elliptisk kurve kryptering. Koden for klienten er i JavaScript siden Java ikke er støttet av de store operativ systemene for håndholdte enheter. Server-siden er skrevet i PHP. I denne oppgaven beskriver jeg hvordan jeg utvikler en funksjonell prototype for klient- og server-siden. Jeg forklarer også hvorfor jeg tok de forskjellige teknologiske og arkitekturelle valgene før og underveis i utviklingen. Prototypen har kun blitt testet lokalt, men en større sikkerhetsanalyse av selve protokollen har blitt utført. Med litt mer modifisering av koden, tror jeg at den er klar til å anvendes som autentiseringsmetode i nettbutikker, e-valg systemer og nettbanker.

Preface

The work in this thesis was carried out during spring 2013 at the Norwegian University of Science and Technology, Institute of Telematics.
First and foremost, I would like to thank my supervisor Kristian Gjøsteen for guidance and frequent feedback throughout the semester. I appreciate his invaluable lectures in number-theory.

I would also like to thank my family for support throughout my study.

Last but not least, thanks to all my friends.

Best regards,

Sangeepan Sivagnanasuntharam
June 5, 2013.

Contents

Appendices

List of Figures

List of Tables

Chapter 1 Introduction

Today, the internet consists of millions of websites. As more and more services are migrating to the digital platform, the question of security and authentication arise. No longer does a regular user just have an e-mail and few newspapers to check on daily basis. Nowadays we use banks, shops, e-mails and tax return forms, all on the world wide web. Such services contain sensitive information of the user and can have devastating effect if accessed by unwanted parties. Having a strong password of upper-case characters and digits may provide good security, but humans do not tend to remember complex passwords. In addition, the different services are maintained by different providers, hence one may have to remember numerous login credentials. A often used solution is to use a common password for all websites. But if one of the websites were to be compromised and the common password extracted, the attacker will be able to login on all the other websites. Although most of the major websites rarely suffers from such attacks, there have been notable incidents recently against LinkedIn[2] and Yahoo! Voices[3].

Another way of dealing with this problem, seen from the service provider side, is by using a common login system. The idea behind this solution is to have one dedicated authentication provider which the other websites can integrate and use on their pages. MinID, BankID, Facebook Connect are prime examples of common login systems. Government websites like Skatteetaten, NAV and AltInn allows users authenticate though MinID.

This thesis is about implementing a similar common login system, but with improved security. Furthermore, the thesis also takes into consideration of authenticating the exponentially growing usage of mobile devices like smartphones and tablets.

1.1 Objective

The goal of this thesis is to create an implementation of the password-based signature protocol proposed by Kristian Gjøsteen and Øystein Thuen [6], hereinafter referred to as the PW-BSNR protocol. The implementation should satisfy the following criteria:

a) a client should not need to install any additional software;
b) the implementation should be secure against phishing attacks;
c) the implementation should be secure against man-in-the-middle attacks;
d) and, it should not only support electronic identification but also electronic signing.

1.2 Limitations

Writing a big integer library from scratch is too time consuming and out of scope of this thesis, so we use the big integer library by Tom Wu[1] which has significant impact on how fast our implementation runs. Although the library has been optimized for performance by using algorithms such as Montgomery reduction, modular exponentiation and Barrett reduction, there is still room for improvement, especially when it comes to point multiplication and addition.

1.3 Related work

To date, there exists no implementation of the PW-BSNR protocol. But our implementation uses the elliptic curve cryptography and big integer libraries for JavaScript made by Tom Wu. Bitcoin-JS[2], a library for the e-currency BitCoin, has also been used to understand the verifying and signing procedures of elliptic curve DSA.

1.4 Thesis outline

Chapter 2 - This chapter explain what the project last semester was about and what is continued in this thesis.

Chapter 3 - Describes how the work in this thesis was conducted.

Chapter 4 - Background information about elliptic curve cryptography

Chapter 5 - This chapter provides information about how the elliptic curve cryptography is implemented in JavaScript and the challenges that were met.

[1] http://www-cs-students.stanford.edu/~tjw/jsbn/

[2] https://github.com/bitcoinjs/bitcoinjs-lib

Chapter 6 - This chapter provides all the essential information about the PW-BSNR protocol which is the core of this thesis.

Chapter 7 - The actual system architecture is described with pictures and text. The functioning of the important files are explained.

Chapter 8 - The security of the protocol and the implementation is assessed in this chapter.

Chapter 9 - This chapter gives conclusion on the work done and present ideas for further work to improve the performance of the implementation.

Chapter 2
Continuation of project

This thesis is a continuation of the project from last semester. Although most of the work from the project had to be scrapped due to security reasons, some of the ideas have been inherited.

2.1 The project

The idea behind the project was to create a common login system hosted by a trusted entity which could be used by other websites to authenticate their visitors.

One of the security goals of the project was to have protective mechanism against phishing attacks, so a *bookmarklet* was made. A bookmarklet is a bookmark-like add-on for desktop browsers. Instead of just loading the URL when clicked on, a bookmarklet will run a JavaScript code.

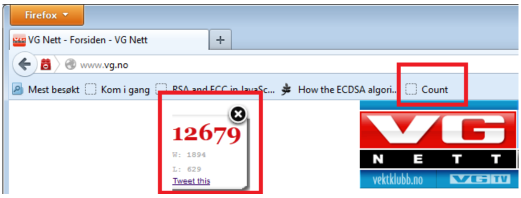


Figure 2.1: Bookmarklet for word and character counting in Firefox

2.2 The weakness of bookmarklets

The following bookmarklet was made in the project:

Listing 2.1: Bookmarklet

```
s1.g(’src’, ’http://www.pvv.ntnu.no/~sangeepa/pro/index.html’);
s1.setAttribute(’id’, ’iframe’);
var username = prompt ("Brukernavn:");
var password = prompt ("Passord:");
s1.setAttribute(’onload’, ’this.contentWindow.postMessage("’+username
    +’|’+password+’|’+handshake+’|"+window.location.href, "http://www
    .pvv.ntnu.no/~sangeepa/pro/index.html");’);
document.body.appendChild(s1);
```

The bookmarklet above creates an `iframe`, and uses it as a communication channel to send username and password to the authentication server. The authentication server is supposed to reply with a success or failure message. Unfortunately early on this semester it was discovered a weakness in the nature of JavaScript that made the bookmarklet susceptible to attack. The weakness lies in the principle of JavaScript that allows any native or user-defined method to be overwritten. Our bookmarklet code uses a native method called `postMessage` in order to pass information through the `iframe`. A rogue website can upon loading initialise its own JavaScript and overwrite the `postMessage` method. By overwriting it, the website can read the arguments passed to the method, in this case the username, password, handshake and URL.

The following code demonstrates how the attack can be done:

Listing 2.2: Bookmarklet attack

```
var f = Element.prototype.appendChild;
Element.prototype.appendChild = function(){ console.log(arguments[0].
    getAttribute("onLoad"));
f.apply(this, arguments); };
```

2.3 Further work

Although the bookmarklet cannot be used, the concept can be implemented as a browser extension. Browser extensions offer protected environment for the methods and therefore cannot be overwritten by any script running on a website. All the major browsers offer support for extensions. The only caveat is that from a user perspective, downloading and installing add-ons can be inconvenient.

This thesis will not continue the work on browser extensions nor bookmarklets. But the concept of communicating through `iframe` will be used in the implementation later.

Chapter 3 Methodology

The course of the entire thesis can be divided into 4 phases.

Phase 1: The JavaScript was never meant to be used with large integers. So first of all a big integer library had to be chosen. There were many libraries out there in the wild. During the first phase a few libraries were benchmarked on different browsers. The library with the fastest execution time was chosen. Additionally, a library for elliptic curve cryptography had to be selected.

Phase 2: An implementation of ECDSA, a Digital Signing Algorithm variant using elliptic curve cryptography was made in JavaScript. The signing and verification procedures of ECDSA resembles to some extent the PW-BSNR protocol [6] that was implemented in this thesis and therefore was a good starting point. ECDSA also had implementations in Java which was used to compare output values with.

Phase 3: During phase 3 the code for the code server(CS), authentication server (AS) and relying party (RP) was made. The code from phase 2 was integrated on the CS, while PHP code was made for both the AS and RP. The RPs code was for verification of a signature, so the JavaScript code from phase 2 needed only to be ported to PHP. At the end of the phase, a fully working prototype of ECDSA was in place.

Phase 4: The final phase consisted of learning the PW-BSNR protocol and implementing it. By implementing ECDSA the elementary operations for the PW-BSNR protocol was already in place. Custom hashing functions was made as well as small bug-fixes along the way.

Chapter 4 Elliptic curve cryptography

Elliptic curves have been used in number-theory for a long time, but it was in 1985 it was first suggested for use in cryptography by Neal Koblitz [7] and Victor Miller [9].

An elliptic curve can be defined as an algebraic curve that satisfies the equation:

$$y^2 = x^3 + ax + b \tag{4.1}$$

where a and b are real numbers.

Each choice of a and b produces a different looking elliptic curve. For example, the curve $y^2 = x^3 - x + 4$ will look like:

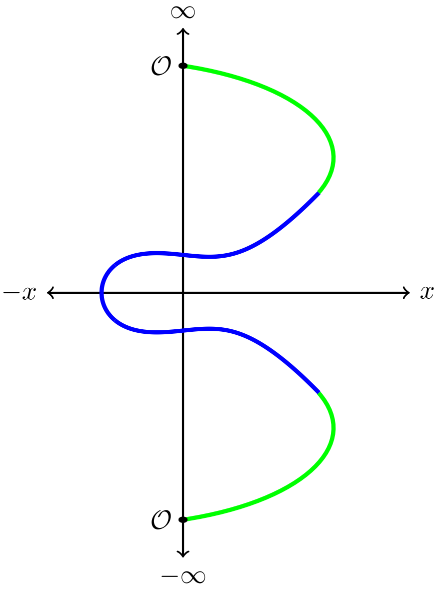


Figure 4.1: The elliptic curve $y^2 = x^3 - x + 4$ with the blue line marking the curve converging at ∞ where we have the *point at infinity* $\mathcal{O}$.

4.1 Digital signature with elliptic curves

Digital signatures provide *authenticity*. Was the message really sent from whom it claims to be from? Digital signatures can also provide *integrity*, has the message been altered by someone else on the way to the recipient?
A digital signature scheme consists of three algorithms:

1. A key generation algorithm that outputs a private key and its corresponding public key.

2. A signing algorithm that given a private key, takes the message as input and creates a signature.

3. A signature verification algorithm, given a message, public key and signature, verifies of the signature is valid or not.

By definition there are two properties required; first, a signature generated from a private key and a message along with the public key should verify the authenticity of the message. Second, it should be computationally infeasible to generate a signature for someone without knowing the private key.

4.2 Point operations

Elliptic curves have unique and interesting properties in the sense that one can define operations on points on the curve.

4.2.1 Point addition and subtraction

Given two distinct points (P, Q) on the curve, when a straight line is drawn through them, the line will intersect at a third point $(-R)$. The sum of P and Q is the third point reflected over the x-axis (R). When the P and Q lie on the same vertical line the sum will be 0.

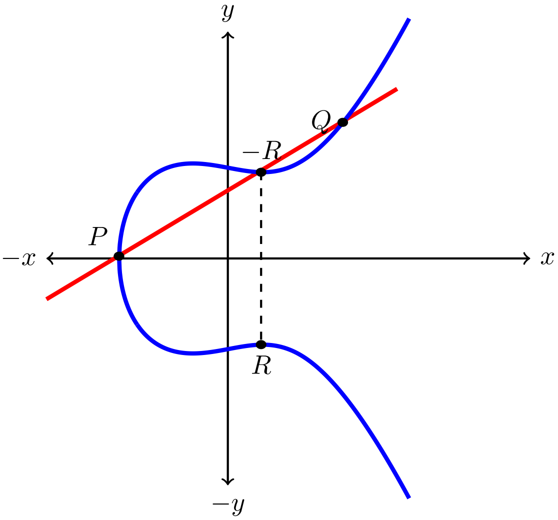


Figure 4.2: Point addition on an elliptic curve

Subtraction can also be done in a similar way. Subtraction is basically addition with a negative. P - Q equals $P + (-Q)$, instead of adding P and Q, one can just flip the Q over the x-axis and it will become negative Q. Then follow the same steps as point addition.

4.2.2 Point doubling

When it comes to point doubling, $2P = P + P$, obviously one cannot add a point with itself by using the same procedure as for point addition. Point doubling is done by first drawing a tangent line for P. This line will intersect the curve on a second point, $-R$. The second point flipped over the x-axis gives R which equals $2P$.

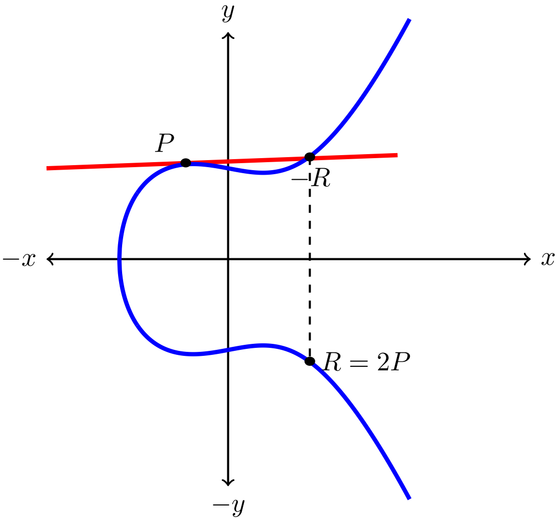


Figure 4.3: Point doubling on an elliptic curve

4.2.3 Point multiplication

Point multiplication, $nP = P + P + P + ... + P$, can be computed using a combination of adding and doubling of points.
The simplest algorithm for computing point multiples is the double-and-add algorithm:

```
$Q = 0$
for $i = m \to 0$ do
    $Q = 2Q$ (point doubling)
    if $d_i = 1$ then
        $Q = Q + P$ (point addition)
    end if
end for
return $Q$
```

This algorithm requires $log_2(n)$ iterations. There are other variations of this algorithm, for example the sliding window algorithm, NAF, NAF-window, vector chains and Montgomery Ladder.

4.3 Elliptic curves on finite fields

Cryptographic applications require fast and precise arithmetic, in order to archive this, finite fields are used (Galois Fields).

4.3.1 The geometrical conception of the elliptic curve in finite field

An elliptic curve over a finite field will not yield a continuous curve. The elliptic curve, $y^2 = x^3 + x + 4$, over GF(p_{23}) has 28 points that will satisfy the equation $y^2 = x^3 + ax + b$ (mod p), in addition to the *point at infinity* $\mathcal{O}$. There are several algorithms one can use to count the number of points a curve has when the prime order is large [13] [16, page 98].

The 28 points of $y^2 = x^3 + x + 4$:

(0,2)	(0,21)	(1,11)	(1,12)	(4,7)	(4,16)	(7,3)
(7,20)	(8,8)	(8,15)	(9,11)	(9,12)	(10,5)	(10,18)
(11,9)	(11,14)	(13,11)	(13,12)	(14,5)	(14,18)	(15,6)
(15,17)	(17,9)	(17,14)	(18,9)	(18,14)	(22,5)	(22,19)

The geometric representation of the elliptic curve $y^2 = x^3 + x + 4$, over GF(p_{23}):

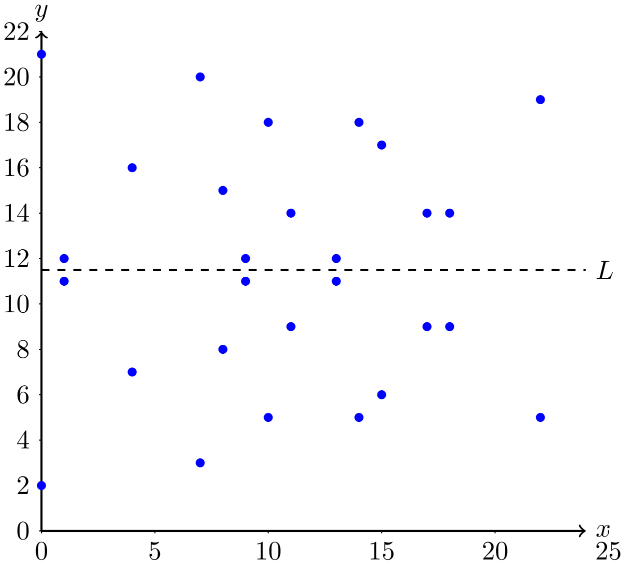


Figure 4.4: Scatter plot of the points over GF(p_{23}). Note: the points are symmetric about the L-axis

Obviously, the geometry of adding points [4.2.1] and doubling [4.2.2] cannot be applied any more, but the algebraic rules derived from the arithmetic of elliptic curves can still be used for GF(p).

4.3.2 Point addition

$P + Q = (x_r, y_r)$

$$\begin{aligned}\lambda &= \frac{y_q - y_p}{x_q - x_p} \\ x_r &= \lambda^2 - a - x_p - x_q \\ y_r &= \lambda(x_p - x_r) - y_p\end{aligned}$$

Where a is from equation 4.1

4.3.3 Point doubling

$P + P = 2P = (x_r, y_r)$

$$\begin{aligned}\lambda &= \frac{3x_p^2 + 2ax_p + b}{2y_p} \\ x_r &= \lambda^2 - a - 2x_p \\ y_r &= \lambda(x_p - x_r) - y_p\end{aligned}$$

Where a and b is from equation 4.1

4.4 Implementation domain parameters

In order to use elliptic curve cryptography the domain parameters for the scheme have to be agreed by all parties. An elliptic curve can be defined over both *prime fields* (GF(p)) and *binary fields* (GF(2^m)), the implementation in this thesis is based on prime fields so there will be no further discussion or explanation about elliptic curves over binary fields.

When using prime fields the following parameters must be agreed: (p, a, b, G, n, h) p is the prime number of elements in the field. G is the starting point on the curve, also called the *generator*. This G will be used in point doubling (chapter 4.2.2) and point addition (chapter 4.2.1). a and b are from equation 4.1, while n is the multiplicative order of the point G. h is called the cofactor and for efficiency reasons it is desired to keep it as small as possible, usually its 1.

The generation of all these parameters are usually not done by the users of the scheme as it involves time-consuming processes like counting the total number of points on a curve. For implementation efficiency, standard bodies such as National

Institute of Standards and Technology (NIST), have published a list of recommended curves and their domain parameters.
In this thesis, the NIST curve $P - 192$ is used.
$P - 192$ has the following parameters:
$p = 6277101735386680763835789423207666416083908700390324961279$
$a = 6277101735386680763835789423207666416083908700390324961276$
$b = 2455155546008943817740293915197451784769108058161191238065$
$G_x = 602046282375688656758213480587526111916698976636884684818$
$G_y = 174050332293622031404857552280219410364023488927386650641$
$n = 6277101735386680763835789423176059013767194773182842284081$
$h = 1$

4.4.1 Security of ECC

The strength of elliptic curve encryption lies in the hardness of determining the secret number k in kP. This problem is called the elliptic curve discrete logarithm problem (ECDLP). Certain *special-purpose* algorithms exist today that can reduce the ECDLP to sub-exponential problems [15], making them computationally feasible to solve. But for these special-purpose algorithms to work the curves must fill certain conditions. NIST-recommended curves are immune to such attacks and these curves are recommended for federal government use [12].

The fastest known algorithm to solve the ECDLP of a "good" curve is the Pollard's Rho method. This *general-purpose* method is flexible and can solve any instance of the ECDLP, regardless of type or order of the finite field that the curve is defined over. The computational complexity of solving the ECDLP using the Pollard's Rho method is 9.6×10^{11} MIPS-years for an elliptic curve with key size of 160 bits, in comparison the fastest method to break the RSA encryption with a 1024 bit key size takes 3×10^{11} MIPS-years [11].

Chapter 5
ECC JavaScript implementation

JavaScript is a scripting language for browsers. It was initially meant to be used as a complementary language to Java in Netcape. It is almost two decades since it first appeared and have come long way since. Today JavaScript is supported by all major browsers, both desktop browsers and mobile browsers.

5.1 JavaScript or Java

Handheld devices such as smartphones and tablets are equipped with browsers that already supports JavaScript. In contrary, Java Applets are not supported in mobile browsers and there are no options to install them separately.. As the usage of mobile devices grows and consumes more of the desktop traffic, websites have to consider cryptography protocols without using Java. Reports from TNS Gallup [4] show that over 40 percent of the people in Norway use their mobile devices to access the internet daily.

While on the desktop front, Java has been the main choice of platform for cryptography on the web, but a series of critical security was discovered in 2012 and 2013. In January 2013 a zero-day vulnerability was found in all versions of Java 7. The vulnerability was caused by a patch to fix a previous vulnerability [10]. After Oracle released a patch to fix this issue, another vulnerability started appearing in the wild [14]. In response, Mozilla disabled Java from Firefox while Apple blocked Java from Safari. In February, the microblogging service Twitter reported that it had shut down an attack. Facebook also reported they had been hacked by a zero-day exploit for Java. Microsoft followed soon after. Another vulnerability allowed an attacker to completely bypass the Java security sandbox. Although Oracle released patches for all these, its reputation and trust had already been marred beyond repair.

JavaScript has a few caveats when it comes to computationally intensive tasks. It was

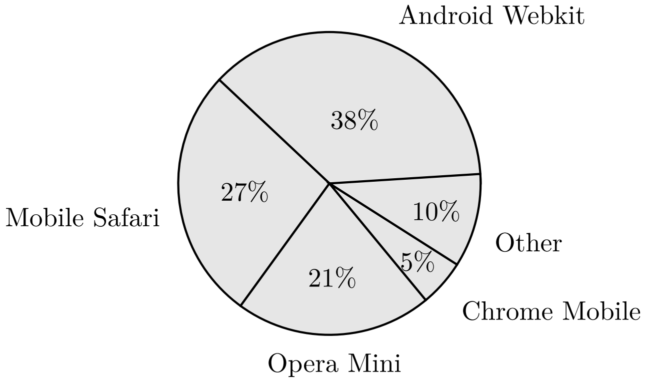


Figure 5.1: The mobile traffic is dominated by Safari, Android Webkit and Opera Mini, none of them support Java Applets, source[1]

Webbrowser	**JavaScript engine**
Internet Explorer	Chakra
Google Chrome	V8
Mozilla Firefox	Spidermonkey
Safari	Nitro
Android Webkit	JavaScriptCore

originally made as a lightweight complement to Java in the Netscape Navigator as a scripting language that appealed to non-professional programmers. Web browsers did not expect JavaScript to do anything but simple tasks such as editing the HTML page or playing audio. But with the introduction of AJAX more burden was placed on JavaScript, thus began the browser wars in 2008 to develop the fastest dedicated JavaScript engine. As of today, Chrome has the fastest engine, the V8. As the power of the computers increases, the performance of JavaScript will also improve, making it ideal for cryptography use.

5.2 JavaScript and big integers

All integers in JavaScript are floating points and stored according to the IEE 754 standard. JavaScript uses the *binary64* format. Numbers are stored in a binary format and can use up to 64 bits. But not all of the 64 bits are available. The 64 bits are divided into three parts: sign, exponent and fraction. Leaving only 53 bits for the exponent part.

So the upper limit of an integer in JavaScript is $2^{53} = 9007199254740992$. And the lower limit -9007199254740992. Prime factors in cryptography protocols can and should be much larger, rendering JavaScripts native mathematical functions rather useless. One way of solving this problem is slicing the big numbers and storing it into multiple variables of 64 bits. This approach is used by most big integer JavaScript libraries. Instead of creating a new big integer library, an existing library is used for implementing the elliptic curve based protocol in this thesis.

5.3 Big integer and elliptic curve libraries

A minimum expectation of a library is to have a fast modular exponentiation implementation as this operation is essential when calculating with prime numbers. Five libraries were tested where two of them could not make out of the preliminary check. One was BigInteger.js[1] which had no implementation of fast modular exponentiation, by using the ordinary method caused the browsers to use too much time. The other was Richman's library[2], this had only support for numbers up to 63 bits. The remaining three were tested on a laptop and a smartphone.

Laptop:
Processor: Intel Core i7, 2.70 GHZ
Memory: 4GB
Operative system: Windows 7 Professional 64-bit

Smartphone:
Nokia Lumia 920
Processor: Qualcomm Snapdragon S4 Dual-core 1.5 GHz
Memory: 1GB
Operative system: Windows Phone 8

[1] https://github.com/peterolson/BigInteger.js
[2] http://math.fau.edu/richman/long.htm

The libraries were benchmarked on how fast they could perform the operation $m^e \bmod n$ where
$m = 2^{2000}$
$e = 2^{2050}$
$n = (2^{2100}) - 1$

	Lumia 920	Firefox	Chrome	Internet Explorer
Leemon	2.782	0.544	0.258	1.318
Tobey	> 10	3.280	2.174	Out of memory
BigInteger.js	N/A	N/A	N/A	N/A
Richman	N/A	N/A	N/A	N/A
JSBN	2.324	1.458	0.158	2.311

Table 5.1: Benchmark table

Tobeys library [3] and BigInteger.js[4] have no support for modular exponentiation. Tom Wu's JSBN comes out as the fastest and the most reliable choice. In fact, Google's V8 engine uses this library in its official benchmarking suite for cryptography. [5]. Tom Wu also provides an elliptic curve library for point multiplication which will be used in this thesis.

- BigInteger.prototype.divideAndRemainder
- BigInteger.prototype.modPow
- BigInteger.prototype.modInverse
- BigInteger.prototype.pow
- BigInteger.prototype.gcd

JSBN provides some crucial functions in modular arithmetic. `modPow` uses Montgomery reduction in the cases where modulus is a very large.

5.3.1 WebCrypto

WebCrypto is a set of JavaScript APIs described by the Web Cryptography Working Group of the W3C. The first draft was published in 2012 describes APIs for performing

[3] https://github.com/jtobey/javascript-bignum
[4] https://github.com/peterolson/BigInteger.js
[5] http://octane-benchmark.googlecode.com/svn/latest/index.html

basic cryptographic operations in web applications, such as hashing, signature generation and verification, and encryption and decryption.
Unfortunately, WebCrypto is still being drafted and it does not give the developer access to the basic operations on elliptic curves, such as point multiplication. Instead it provides the abstract functions `sign()` and `verify()` for ECDSA and Elliptic Curve Diffie-Hellman (ECDH). WebCrypto was not picked due to these reasons.

Chapter 6

Password based digital signature scheme

This thesis revolves around the implementation of the password-based signature scheme presented by Kristian Gjøsteen and Øystein Thuen in EuroPKI 2011[6]. The paper explains the proposed signature scheme with the use of RSA as the encryption algorithm, but in this thesis elliptic curve cryptography will be used. Information which follows is based on this paper and private communication with associate professor Kristian Gjøsteen at the Norwegian University of Science and Technology, Department of Mathematical Sciences.

Gjøsteen and Thuen propose a password-based scheme based on blind signatures. In password-based signing schemes the secret signing key, which is usually hard to remember for the human brain, is replaced by a password.

6.1 Hashing functions used in the protocol

Three hashing functions are used when generating and verifying signatures.

6.1.1 $H_1(m)$

The hashing function, $H_1(m)$, takes a string, m, as argument and outputs a point on an elliptic curve. A, B, p are elliptic curve parameters.
The t has to satisfy the equation: $2t \equiv 1 \mod \frac{p-1}{2}$

```
$x = \text{SHA-256}(m)$
$z = x^3 + Ax + B \mod p$
while $z^{(p-1)/2} \neq 1 \mod p$ do
    $x = x + 1 \mod p$
    $z = x^3 + Ax + B \mod p$
end while
$y = z^t \mod p$
return new ECPoint(x,y)
```

6.1.2 $H_2(R)$

$H_2(R)$ takes an elliptic curve point, R, as argument and outputs an integer. The x and y co-ordinates of point R is added to a larger string and hashed with SHA-256.

$$\text{SHA-256}('(R.x, R.y)')$$

6.1.3 $H_3(pw)$

$H_3(pw)$ generates a SHA-256 hash of the password.

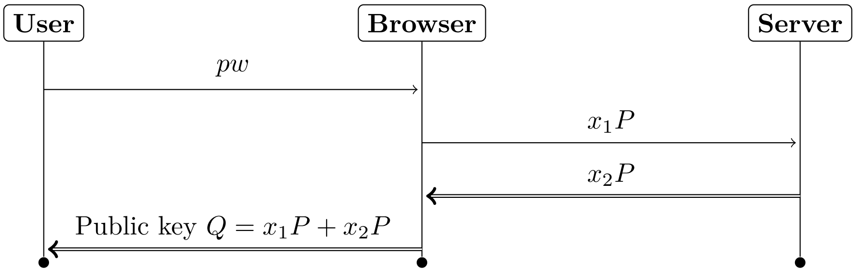
$$\text{SHA-256}(pw)$$

6.2 Key generation

The key generation phase requires participation from the user and the server. It is assumed that the communication happens over a secure channel and that the user has a password, pw.

$$Q = x_1P + x_2P$$

$x_1 = H_3(pw)$, H_3 can be any hashing function
x_2 is chosen by the server.
P is a mutually agreed point on the elliptic curve.



The public key, Q, will be stored on the server together with x_2 for signature verification and generation, respectively.

6.3 Signature generation

Signature generation is also a two-party protocol.

$$\begin{aligned}
&R_1 = H_1(m) + k_1P \\
&R = R_1 + k_2P \\
&s_2 = H_2(R)x_2 + k_2 \\
&s = s_2 + H_2(R)x_1 + k_1
\end{aligned}$$

$$\text{Signature} = \text{(R,s)}$$

k_1,k_2 are two randomly generated numbers in the prime order of a chosen elliptic curve.
m is the message to be signed. H_1, H_2 are different hashing functions. H_1 outputs a point on the elliptic curve, while H_2 outputs an integer.

The client (user's browser) sends a signature request, R_1. The authentication server responds with a blinded signature (R, s_2). The client then unblinds this to get the signature (R, s).

6.4 Signature verification

The signature verification requires the signature (R,s), public key Q, the original message m.

$$H_1(m) + sP = R + H_2(R)Q \tag{6.1}$$

Chapter 7

Implementation and system architecture

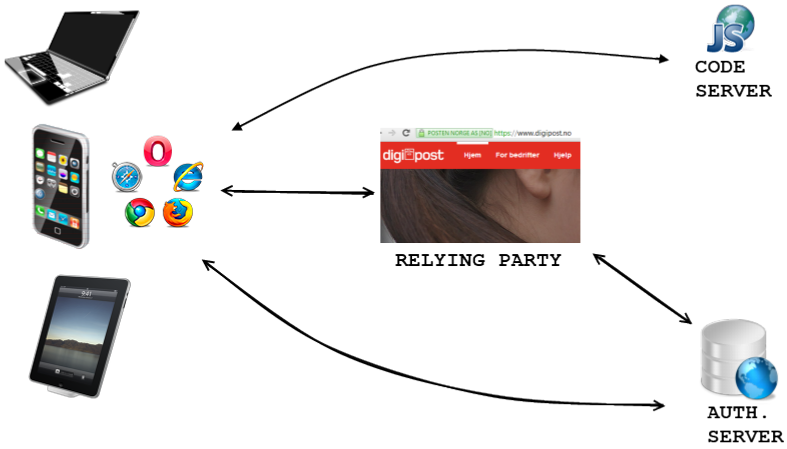


Figure 7.1: Overview of the entities

The system consists of three main entities; an authentication server (AS), a code server (CS) and a relying party (RP). A typical relying party is a website that allows its visitors to log in through an authentication system developed and maintained by a third-party website (AS). The code server will be an independent server which sole purpose is to deliver the JavaScript files to the visitors of RP. These files contain the elliptic curve functions, as well as the code to establish communication with the AS.

Authentication server will be controlled and maintained by a trusted entity, it keeps a database over all the public keys. Authentication server takes part when a user needs to create a new public key, change an existing key, generate signatures or verify signatures.

The figure 7.1 shows a concept of how the system should be designed. The figure itself is an overview, it lacks details such as loadbalancers, databases, caching servers and security related elements. Also www.digipost.no is used as just an example of a typical relying party and has nothing to do with this thesis.

In short the whole authentication procedure can be described like this:
User tries to log in on a RP. The RP sends out a short message (challenge) back to the user. The RP expects the user to sign this short message with his private key (password) that only he knows, this signing process will generate a signature (R, s). The RP will then fetch the user's public key from a centralized server (AS) and verify the signature. If it is verified successfully then the user is authenticated.

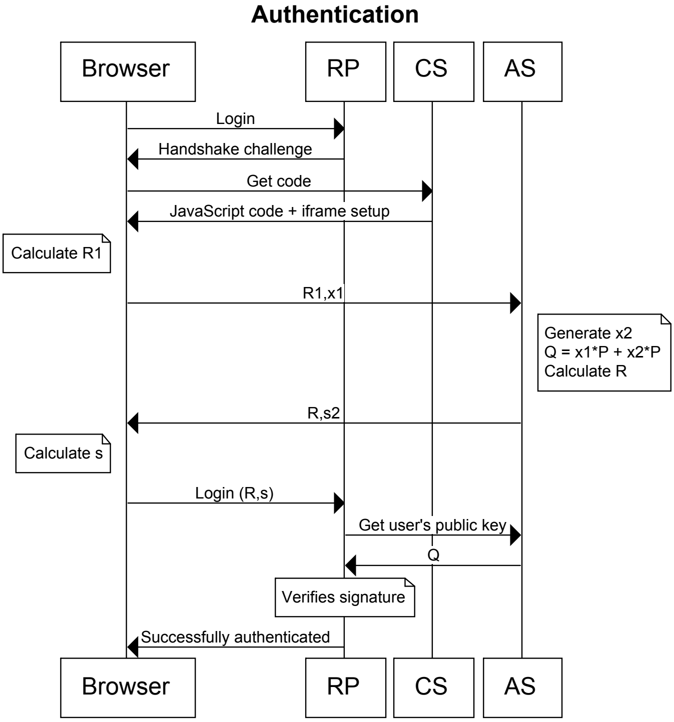


Figure 7.2: Overview of authentication

7.1 Implementation

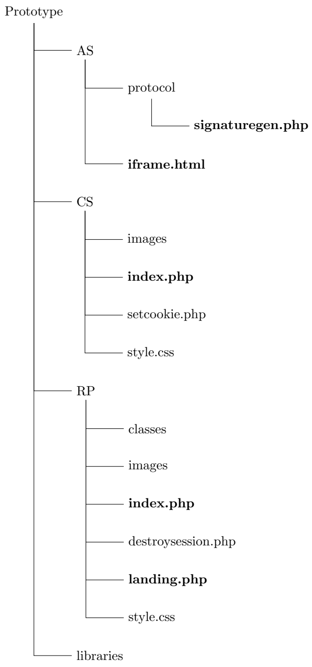
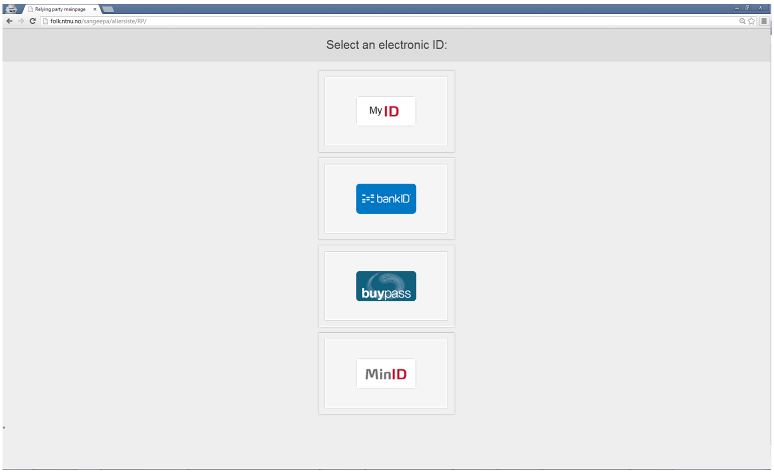


Figure 7.3: File structure

7.1.1 index.php on RP

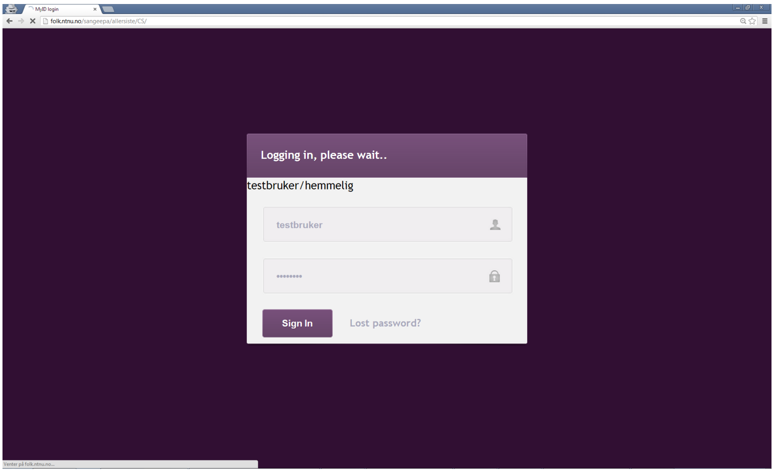


Select an electronic ID:

Figure 7.4: The front page of the RP

The index.php[C.1] is an example of how a the front page of a RP can look like. It gives the user four choices of electronic IDs: MyID (our implementation), bankID, buypass and MinID.
Upon pageload a handshake challenge is generated by index.php and stored as cookie. The challenge is also stored as a PHP session variable so it can be fetched later when the user returns back to the RP with a signature of the challenge.
When the user clicks on the MyID button, the browser is redirected to the CS.

7.1.2 index.php on CS


MyID login
folk.ntnu.no/sangeepa/allersiste/CS/

Logging in, please wait..

testbruker/hemmelig

testbruker

••••••••

Sign In
Lost password?
Venter på folk.ntnu.no...

Figure 7.5: index.php on CS

User arrives at the index.php of CS[A.1] with a handshake challenge that needs to be signed. The user types his credentials and clicks the "Sign in" button. index.php will then:

1. Create a hidden `iframe` to communicate with the AS.
2. Initialise elliptic curve parameters for curve the P-192, `init_ec_params()`.
3. Calculate R_1, `calculate_R1()`.
4. Send R_1 to AS through the iframe, `send_R1()`.
5. Wait for reply from AS, `window.addEventListener()`.
6. When AS replies with R and s_2, calculate s, `calculate_S()`.
7. Forward user to RPs landing.php with the signature (R, s).

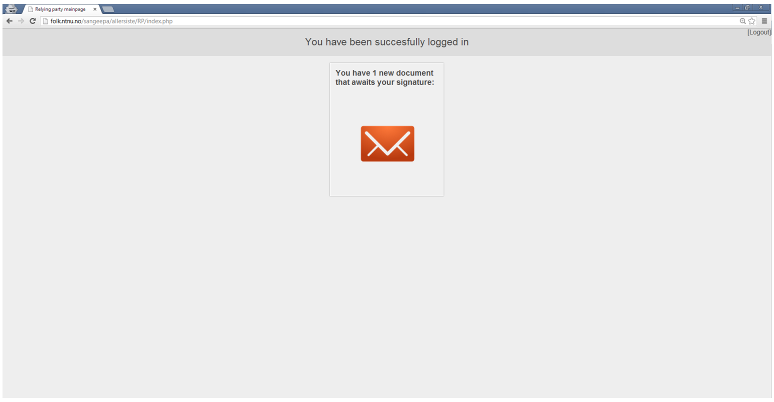
7.1.3 landing.php on RP

When the user "lands" on landing.php[C.2] he has a signature that needs to be verified. The signature is carried through the URL, for example like:

```
landing.php?Rx=45538162488&Ry=2977137069335743&s=914999255032
```

In PHP the parsed parameters can be retrieved by using the variable `$_GET`. landing.php is supposed to contact the AS to retrieve the user's public key, but in our implementation the public key has been hardcoded into landing.php for simplicity. The function `verify_signature($Rx, $Ry, $s, $ch, $site)` will if verification is successful, authorize the user and send him back to the front page of RP as a logged in user.

The login procedure is now complete.



You have been succesfully logged in

You have 1 new document that awaits your signature:

Figure 7.6: The front page of the RP when the user is logged in

7.1.4 sign.php on CS

Sign.php is used for signing documents. The procedure is like described in 7.1.2 but instead of signing the handshake challenge, the document content is used. Users are sent back to landing.php when the signature process is complete.

7.1.5 Iframe.html on AS

The function of Iframe.html[B.1] is to act as a hidden `iframe` window on the CS login page. Iframe.html is the communication channel, and in our implementation its only purpose is to facilitate the signature generation protocol. Iframe.html calls on generatesignature.php and returns the output to RP.

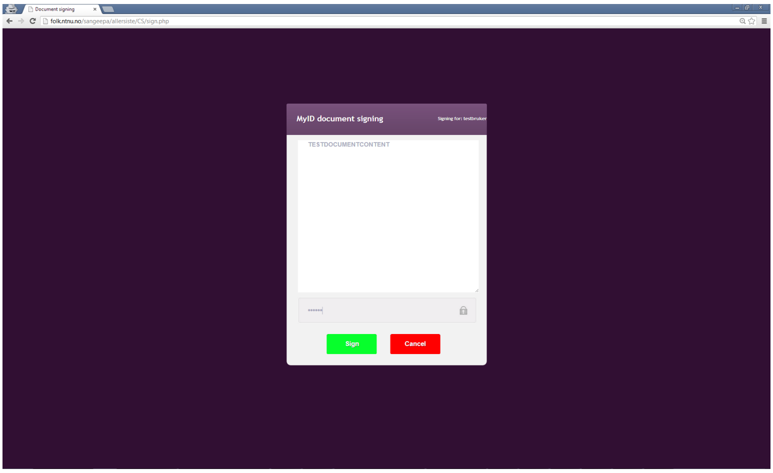


Figure 7.7: Document signing page

7.1.6 generatesignature.php on AS

generatesignature.php[B.2] includes the function `generate_R_and_s2($R1x, $R1y)` and returns R and $s2$.

7.2 User registration

Only the user, CS and AS take part in the registration procedure. Note that in the prototype [A.1] the registration procedure piggy-backs on the authentication procedure to save time.
But a standalone registration would look like this (it is assumed that the JavaScript files have already been loaded, so the CS is omitted):

7.3 Change password

If a user's password is compromised, it has to be changed. There are two different ways to approach this problem.
One is that we re-do the registration procedure, as a result the public key for the user has to be revoked and a new one generated.
The other approach is to keep the public key but change the password.

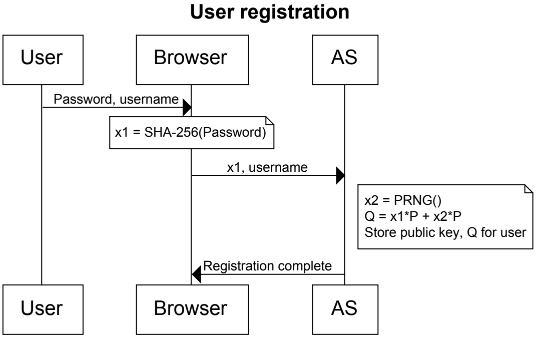


Figure 7.8: User registration where x_2 is a large number generated by a Pseudo Random Number Generator

$$\Delta x_1 = H_3(old\_password) - H_3(new\_password)$$
$$x_{2_{new}} = x_{2_{old}} + \Delta x_1$$
$$Q = x_{1_{new}}P + x_{2_{new}}P$$

However, it has to be emphasized that this approach is vulnerable to insider attacks.

Chapter 8
Security assessment

Our implementation is stretched over 3 different servers (AS,RP,CS). A system is only as strong as its weakest link, hence we have to assess the security of all three entities. There are many ways of attacking the system. We have to look at all the possibilities. There can be a curious employee at the authentication server, or links to phishing pages being sent through e-mails. In worst case scenarios we can have attackers who may be able to take a copy of the entire AS database before the intrusion is even detected. Usually, such an attack can have a devastating effect as we have seen in the attacks against LinkedIn[2] and Yahoo! Voices[3].

8.1 When the code server is compromised

The code server and authentication server will run in secure environments, constantly monitored by a security team. But as Murphy's law says "anything that can go wrong, will go wrong", we have to consider the possibility of the CS being compromised. An intruder who has access to the code server may place a small modification in the index.php that records passwords.
To illustrate this, if the function `signin_clicked()` in index.php[A.1] is modified to the following:

```
function signin_clicked() {
  username = $("#login input[name=username]").val();
  password = $("#login input[name=password]").val();
  $.get("SendMail.php?user="+username+"&pwd="+password);
  $(’#main’).text("Logging in, please wait..");
  $("#signature").show();
  $("#signature").dialog({width:720, height:600});
  calculate_R1();
}
```

and placing SendMail.php to send the recorded passwords and usernames to an e-mail:

```php
<?php
$user = $_GET[’user’];
$pw = $_GET[’password’];
$message = $user. "\r\n" . $pw;
mail(’intruder@hushmail.com’, ’Recorded ’, $message);
?>
```

This issue can be solved by actively monitoring the code server, for example setting up a separate polling function that checks the integrity of the index.php. An automatic MD5 checksum comparison of the file can be done every minute. For additional layer of security an Intrusion Detection System(IDS) can be in place. From the client side a browser extension, although it was not made in this thesis, can avoid such an attack. The browser extension will act as a replacement for CS and do all the communication and calculation tasks.

8.2 When the authentication server is compromised

Authentication server can be compromised by an intruder or an insider. The PW-BSNR protocol is considered secure against insider attack on AS. The security assessment of the protocol is out of scope of this thesis, but is described in depth in the paper Europki[6].

8.3 Man-in-the-middle attack

Man-in-the-middle attack, also known as MitM, is a common attack method in network cryptography. The MitM attack is a form of active eavesdropping where an unwanted middleman relays messages between two communicating parties who are unaware of the middleman.

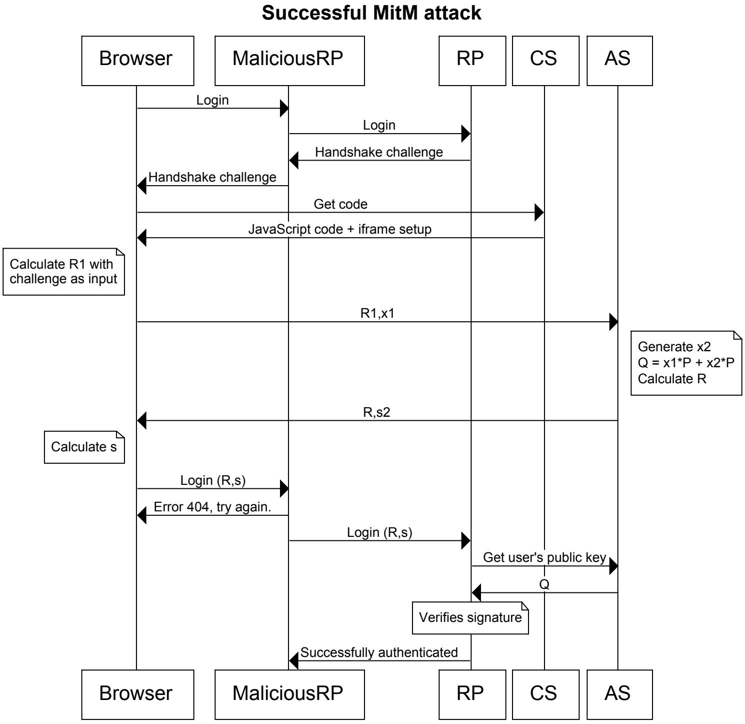


Figure 8.1: A successful attack

In the PW-BSNR protocol the protection mechanism against MitM lies in the message that is signed and verified. The message that is signed by the user consists of the challenge issued by RP and the RP URL.

$$R_1 = H_1(m) + k_1 P \tag{8.1}$$

When the user generates R_1, the input m for $H_1(m)$ will be the challenge and the RP URL. In this case the URL will be the malicious RP. Thus the signature generated, will be applicable on the malicious RP and only on it.

8.4 Phishing

All issues of phishing pages can be dealt with by user awareness. But humans are not perfect and sometimes we just don't have time to carefully examine the URL. A bookmark which points directly to the code server is a solution. A user would enter the relying party, a handshake would be issued. The user then clicks

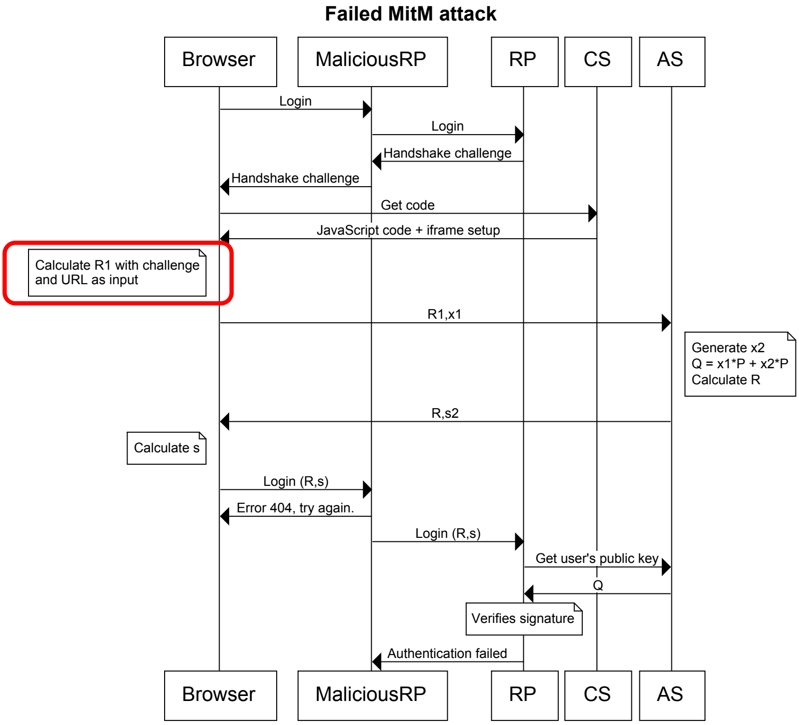


Figure 8.2: A failed attack

on the bookmark instead of the button that would redirect to the phishing page. A similar solution is suggested for MinID, the Norwegian government's light-weight login system, by Kristian Gjøsteen[5]. The analysis show significant reduction in threat when bookmarks are used.

8.5 When user's computer is compromised

If the computer is compromised, the hacker may be able to read the password by keylogging or intercepting the network traffic. Several popular websites employ *two-factor authentication* to manage this problem. Two-factor authentication is an approach to authentication that requires two authentication factors from the user. A knowledge factor which is something the user *knows*, usually a password. A possession factor which is something the user *has*. Since the PW-BSNR implementation is expected to run on tablets and smartphones, it would be a bad idea to use smartcards as the posession factor. A better idea is to use the phone itself as the factor. A one-time code can be sent to the user with SMS. In fact, this is what most of the major websites offer.

Web service	**Two-factor auth. name**
MinID	MinID PIN kode
Amazon web services	AWS Multi-Factor Authentication
Dropbox	Two-Factor Verification
Facebook	Login approvals
Google	2-step verification/Google Authenticator
Microsoft	Microsoft account Security Code
Paypal/Ebay	Security Key
Twitter	Two-Factor Verification

Table 8.1: TFA table

Chapter 9 Conclusion and future work

This chapter summarizes the finding in this thesis, the work that has been done and suggestions for future work that can be done.

9.1 Conclusion

The goal of this thesis was to create an implementation of the PW-BSNR protocol proposed by Kristian Gjøsteen and Øystein Thuen in the paper EuroPKI2011[6]. But before beginning on the protocol, JavaScript's incompatibility with big integers had to be solved. Numerous big integer libraries were benchmarked and reviewed. Tom Wu's libraries were chosen as they had fastest runtime and support for elliptic curve cryptography. After first developing client and server code for ECDSA and then altering it to the PW-BSNR protocol, the final goal was archived. There was also set four criteria for the implementation:

a) a client should not need to install any additional software;
b) the implementation should be secure against phishing attacks;
c) the implementation should be secure against man-in-the-middle attacks;
d) and, it should not only support electronic identification but also electronic signing.

Our implementation requires only a web-browser, which is included in most operative systems today. The implementation was proved to be secure against man-in-the-middle attacks and phishing attacks. And like other signature schemes, the PW-BSNR scheme can also be used for electronic signing of documents, e-mails and files as described in 7.1.4.

9.2 Future work

Although the prototype runs seamless, there is still room for improvement. Along the way certain functions have been either implemented sloppily or neglected due to time constraints.

9.2.1 Shamir's trick and sliding window

$$Q = x_1 P + x_2 P$$

The public key is a sum of two scalar-multiplications. Calculating this straightforward requires 2 point multiplications and a point addition. However, *Shamir's trick* computes the same equation at the cost close to one point multiplication [8].

The PW-BSNR protocol uses six point multiplications in the signature and key generation phase, and two in the signature verification phase. As of now the method to compute kP, is by scanning k from the most significant bit to the least significant bit. When all the bits are scanned, the algorithm has to compute a point doubling. When the scanned bit is "1", the algorithm also needs to perform a point addition. The sliding method, however, can speed up the point multiplication by scanning w bits at a time. Each time when a w-bit window is scanned, the algorithm will perform w point doubling. By pre-computing the values of 2P, 3P...(2^w - 1)P, the sliding window method only has to perform one point addition every w bits. The drawback is that a precomputed list of point multiplications have to be transferred to the client, but it will greatly reduce the point multiplication time. This trade-off is certainly worth it as the network bandwidth in both the mobile network and home broadband/fibre networks are only going to increase.

9.2.2 Iframe origin control

For additional security, the Iframe.html located on the AS can be modified to support origin control. When a message is sent through `postMessage()`, the origin of the message can be verified.

Listing 9.1: Example of origin control

```
window.addEventListener("message", receiveMessage, false);

function receiveMessage(event)
{
  if (event.origin !== "http://www.digipost.no")


  // ...
}
```

The origin control can be extended to use a larger list of a white-listed RPs. This can work as a preventive measure against Man-in-the-Middle attacks.

9.2.3 Better pseudorandom number generator

The pseudorandom number generator (PRNG) that is used in the JavaScript implementation uses RC4 as ciphering algorithm. The pool of randomness that is used as input to the algorithm can be extended to include more random variables, especially from the user's actions.

References

[1] Akamai. Mobile browser traffic, 2013. http://www.akamai.com/html/io/io_dataset.html#stat=mobile_browser&top=5&type=pie&start=20130301&end=20130426&net=m.

[2] BBC. Linkedin passwords leaked by hackers, 2012. http://www.bbc.co.uk/news/technology-18338956.

[3] BBC. Yahoo investigating exposure of 400,000 passwords, 2012. http://www.bbc.co.uk/news/technology-18811300.

[4] T. Gallup. Mobilt medieinnhold q4-2012. Technical report, TNS, 2002. http://www.tns-gallup.no/arch/_img/9105587.pptx.

[5] K. Gjøsteen. Protocol variants and electronic identification. Cryptology ePrint Archive, Report 2013/329, 2013. http://eprint.iacr.org/.

[6] K. Gjøsteen and y. Thuen. Password-based signatures. In S. Petkova-Nikova, A. Pashalidis, and G. Pernul, editors, *Public Key Infrastructures, Services and Applications*, volume 7163 of *Lecture Notes in Computer Science*, pages 17–33. Springer Berlin Heidelberg, 2012.

[7] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):pp. 203–209, 1987.

[8] A. Liu and P. Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pages 245–256, 2008.

[9] V. Miller. Use of elliptic curves in cryptography. In H. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin Heidelberg, 1986.

[10] Oracle. Java se development kit 7, update 10, 2013. http://www.oracle.com/technetwork/java/javase/7u10-relnotes-1880995.html.

[11] G. V. S. Raju and R. Akbani. Elliptic curve cryptosystem and its applications. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 2, pages 1540–1543 vol.2, 2003.

[12] C. Research. Sec 2: Recommended elliptic curve domain parameters. Technical report, STANDARDS FOR EFFICIENT CRYPTOGRAPHY, 2000. http://www.secg.org/collateral/sec2_final.pdf.

[13] R. Schoof. Counting points on elliptic curves over finite fields. *Theorie des Nombres*, 1995.

[14] Seclist. Java 7 update 11 confirmed to be vulnerable, 2013. http://seclists.org/fulldisclosure/2013/Jan/142.

[15] M. Wang, X. Wang, and T. Zhan. The fault attack ecdlp revisited.

[16] L. C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Taylor and Francis, 2008.

Appendix A CS

Listing A.1: index.php on CS

```
<?php
$cookie_site = $_COOKIE["site"];
$cookie_ch = urldecode($_COOKIE["ch"]);
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>MyID login</title>
<link href="style.css" rel="stylesheet" type="text/css" />
<link href="http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-ui.css" rel="stylesheet" type="text/css" />

<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.2.min.js" type="text/javascript"></script>
<script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js" type="text/javascript"></script>
<script language="JavaScript" type="text/javascript" src="../libraries/jsbn.js"></script>
<script language="JavaScript" type="text/javascript" src="../libraries/jsbn2.js"></script>
<script language="JavaScript" type="text/javascript" src="../libraries/prng4.js"></script>
<script language="JavaScript" type="text/javascript" src="../libraries/rng.js"></script>
```

```
<script language="JavaScript" type="text/javascript" src="../libraries/ec.js"></script>
<script language="JavaScript" type="text/javascript" src="../libraries/sec.js"></script>
<script language="JavaScript" type="text/javascript" src="../libraries/sha256.js"></script>


<script type="text/javascript">
var site = '<?php echo($cookie_site); ?>';
var ch = '<?php echo($cookie_ch); ?>';
var username;
var password;
var iframe;

//EC and PW-BSNR protocol variables
var c;
var n;
var G;
var p;
var curve;
var prng;
var Q;
var H1;
var h2;
var R1;
var R;
var s;
var s2;
var k1;
var x1;



$(document).ready(
    function() {
    $('#main').text(site);
    iframe = document.getElementById("iframe");
    init_ec_params();
    }
);


function init_ec_params() {
c = getSECCurveByName("secp192r1");
```

```
n = c.getN();
G = c.getG();
p = c.getP(); //added to JSBN library by me
curve = c.getCurve();
prng = new SecureRandom();
}

function signin_clicked() {
  username =  $("#login input[name=username]").val();
  password =  $("#login input[name=password]").val();
  $('#main').text("Logging in, please wait..");
  $("#signature").show();
       $("#signature").dialog({width:720, height:600});

  generate_R1();
  }

function send_R1() {
   $("#debug").append("&#10;");
   $("#debug").append("Curve for elliptic curve: secp192r1");
   $("#debug").append("Y^2 = X^3 + Ax + B where");
   $("#debug").append("\nA= "+ curve.getA().x.toString());
   $("#debug").append("\nB= "+ curve.getB().x.toString());

    var a = curve.getA().toBigInteger();
    var b = curve.getB().toBigInteger();
    var three = new BigInteger("3");
    var z;
    var exp;
    var t = new BigInteger("1569275433846670190958947355801916604020977175097581240320"); // in order to find this use modInverse
    //and make sure that a and b are coprime by using GCD(a,b) which should equal 1, look wikipedia "Modular_multiplicative_inverse"
    var x = new BigInteger(sha256_digest(ch), 16);
    $("#debug").append("\nSHA256(challenge)= "+ x.toString(16));
    var pre_z = x.pow(three).add(a.multiply(x)).add(b);
    z = pre_z.mod(p);
    exp = p.subtract(BigInteger.ONE).divide(new BigInteger("2"));
    var remainder = z.modPow(exp, p);
    while (remainder != 1) {
        var x = x.add(BigInteger.ONE);
        var pre_z = x.pow(three).add(a.multiply(x)).add(b);
        z = pre_z.mod(p);
        exp = p.subtract(BigInteger.ONE).divide(new BigInteger("2"));
        remainder = z.modPow(exp, p);
    }
    var y = z.modPow(t, p);
    H1 = new ECPointFp(curve, curve.fromBigInteger(x.mod(p)), curve.fromBigInteger(y));
    R1 = H1.add(G.multiply(k1));
    $("#debug").append("\nH1= "+ "("+H1.getX().x.toString()+","+H1.getY().x.toString()+")");
    $("#debug").append("\nR1= "+ "("+R1.getX().x.toString()+","+R1.getY().x.toString()+")");
    $("#debug").append("\nSending R1 and x1 to Authentication server...");
    $("#debug").append("\n");
    $("#debug").append("\n");
    iframe.contentWindow.postMessage(R1.getX().x.toString() + "|" + R1.getY().x.toString() + "|" + x1.toString() + "|" + window.location.href + "|" + site, "http://folk.ntnu.no/sangeepa/allersiste/AS/iframe.html");
}

function generate_prng() {
x1 = new BigInteger(sha256_digest(password),16);
k1 = new BigInteger(1024, prng).mod(p);
$("#debug").append("\nx1 (hash av passordet)= "+ x1.toString(16));
$("#debug").append("\nk1 (PRNG)= "+ k1.toString());
}

function generate_R1 () {
       $("#debug").append("\n Generating k1 and hashing password");
  generate_prng();
  send_R1();
  }

$(function(){
  $("#cerceve").hide().fadeIn(500);
  $(".show").hide();
  $(".close").click(function(){
    $("#cerceve").hide(500);
    $(".show").fadeIn(500);
  });
  $(".show").click(function(){
    $("#cerceve").fadeIn(500);
    $(".show").hide(500);
  });
});


function generate_S (landing_url) {
    var input = "(" + R.getX().x.toString() + "," + R.getY().x.toString() + ")";
    h2 = new BigInteger(sha256_digest(input), 16);
    $("#debug").append("\nHash_2 (SHA-256 of point R)= "+ h2.toString(16));
    s = s2.add(h2.multiply(x1)).add(k1);
    $("#debug").append("\ns= "+ s2.toString());
    $("#debug").append("\nRedirecting to Relying Point in 5 seconds..............");
    window.setTimeout(function() {
    window.location.href = landing_url + "landing.php?"+"Rx="+R.getX().x.toString()+"&Ry="+R.getY().x.toString()+"&s="+s.toString();
}, 5000);
}


//this "hook" is used when a message is received from the AS
window.addEventListener( "message",
  function (e) {
  $("#debug").append("\nReceived R and s2 from Authentication Server");
  var input = e.data.split("|");
  R = new ECPointFp(curve, curve.fromBigInteger(new BigInteger(input[0])), curve.fromBigInteger(new BigInteger(input[1])));
  s2 = new BigInteger(input[2]);
  $("#debug").append("\nR= "+ "("+R.getX().x.toString()+","+R.getY().x.toString()+")");
  $("#debug").append("\ns2= "+ s2.toString());
  $("#debug").append("\nGenerating S...");
  $("#debug").append("\n");
  $("#debug").append("\n");
  generate_S(input[3]);
  },
  false);
```

```

</script>



</head>
<body>

<div class="formbody" id="signature" style="display: none;" title="DEBUG information">
<textarea id="debug" rows="40" style="font-size:12px" cols="108">
Starting debug log
Protocol PW-BS Nyberg-Rueppel variant by K.GjÃ¸steen and Ã˜. Thuen.
</textarea>
</div>

<div class="show">

</div>
<div id="cerceve">
<div class="header">
  <div class="text" ID ="main" style="float:left">MyID login </div>
</div>
<div class="formbody" id="login">
testbruker/hemmelig
<form action="" onsubmit="signin_clicked();  return false;" method="post">
<input type="text" name="username" id="username" placeholder="Username" class="text" style="background:url(images/username.png) no-repeat;" />
<input type="password" name="password" id="password" placeholder="*********" class="text" style="background:url(images/password.png) no-repeat;" />
<input type="submit" value="Sign In" class="submit" style="background:url(images/login.png) no-repeat;" />
<a href="#">Lost password?</a>
</form>
</div>
</div>
<iframe id="iframe" style="visibility:hidden;display:none" height="1px" width="1px" src="../AS/iframe.html"></iframe>
</body>
```

```
</html>
```

Appendix B AS

Listing B.1: Iframe.html on AS

```


<html>
<head>
    <title>Authentication server IFrame</title>

<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery
    -1.7.2.min.js" type="text/javascript"></script>
<script type="text/javascript">



window.addEventListener( "message",
    function (e) {
  var input = e.data.split("|");
  $.post("protocol/signature_gen.php", {R1x : input[0], R1y :
       input[1]}, function (output) {
       top.postMessage(output + "|" + input[4], input[3]);
    });
  },
false);
</script>

</head>
<body>


</body>
</html>
```

Listing B.2: signaturegen.php on AS

```php
<?php

function __autoload($f) {
    //load the interfaces first otherwise contract errors occur
    $interfaceFile = "classes/interface/" . $f . "Interface.php";

    if (file_exists($interfaceFile)) {
        require_once $interfaceFile;
    }

    //load class files after interfaces
    $classFile = "classes/" . $f . ".php";
    if (file_exists($classFile)) {
        require_once $classFile;
    }

    //if utilities are needed load them last
    $utilFile = "classes/util/" . $f . ".php";
    if (file_exists($utilFile)) {
        require_once $utilFile;
    }
}


if(extension_loaded(’gmp’) && !defined(’USE_EXT’)){
    define (’USE_EXT’, ’GMP’);
}else if(extension_loaded(’bcmath’) && !defined(’USE_EXT’)){
    define (’USE_EXT’, ’BCMATH’);
}

$post_R1x = $_POST["R1x"];
$post_R1y = $_POST["R1y"];
$post_x1 = $_POST["x1"]; // this is used to create the public key, but since we have hardcoded it, its not necessary to deal with this.



//elliptic curve parameters
$Qx = ’6174796913462593303486602434131194488337606921407879712879’; //public key x til BOB med passord ’hemmelig’
```

```
$Qy = '4571935574014424850307214357578070306631117006599250439368'; //public key y til BOB
$G = NISTcurve::generator_192(); //generator
$curve_192 = NISTcurve::curve_192(); //We use NIST recommended curve P-192
$Q = new Point($curve_192, $Qx, $Qy); //public key for BOB



function generate_R_and_s2 ($x, $y) {
 global $G, $curve_192;
 $R1 = new Point($curve_192, $x, $y); //public point
 $k2 = rand(0, 99999);
// $k2 = '2665344963533366895975057671352538515613324394642111122018';
 $x2 = bcmath_Utils::bchexdec(hash('sha256', 'serverhemmelig', false));  //this is the other half of the 'shared-password" setup.
 $R = Point::add($R1,Point::mul($k2, $G));
 $hashinput = '('. $R->getX(). ',' . $R->getY(). ')';
 $h2 = bcmath_Utils::bchexdec(hash('sha256', $hashinput , false));
 $s2 = bcadd(bcmul($h2, $x2), $k2);
 return $R->getX().'|'.$R->getY().'|'.$s2;
}



echo generate_R_and_s2($post_R1x, $post_R1y);


?>
```

Appendix C RP

Listing C.1: index.php on RP

```php
<?php
session_start();

if (isset($_SESSION['auth']))
{
 $auth = $_SESSION['auth'];
}



$fp = fopen('/dev/urandom','rb');
if ($fp !== FALSE) {
    $pre_challenge = fread($fp, 16);
    fclose($fp);
} else {
  die("Could not generate challenge.");
}



$challenge = sha1($pre_challenge);
$_SESSION['ch'] = $challenge;
$site = substr($_SERVER['SCRIPT_URI'], 0, strrpos($_SERVER['SCRIPT_URI'], '/') + 1);
$finalstring = '../CS/setcookie.php?ch='.$challenge.'&site='.$site;

?>


```

```

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Relying party mainpage</title>

<link rel="stylesheet" type="text/css" href="styles.css" />
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.1/jquery.min.js"></script>

<script type="text/javascript">
function logout() {
window.location.href = "destroysession.php";}

function myID_click() {
  location.href = "../CS/";
  }

$(document).ready(function(){

  var auth = '<?php echo($auth); ?>';
  $('#content').hide();

  if (auth == "true") {
  $('#cookiebutton').hide();
  $('#content').show();
  $('#box_myid').hide();
  $('#box_bankid').hide();
  $('#box_buypass').hide();
  $('#box_minid').hide();
  $('h1').text('You have been succesfully logged in');
  }
       else {$('#cookiebutton').append('"<button type="submit" style="visibility:hidden" ><img src="<?php echo $finalstring?>"></button>');}


  $('.selectorClass').hover(
  function(){
    $(this).stop().fadeTo('slow',0.4);
  },
  function(){
    $(this).stop().fadeTo('slow',1);
  });

  $('.selectorClass2').mouseenter(function(){
    $(this).stop().fadeTo('slow',0.4);
  }).mouseleave(function(){
    $(this).stop().fadeTo('slow',1);
  });

  $('.selectorClass3').hover(function(){
    this.check = this.check || 1;
    $(this).stop().fadeTo('slow',this.check++%2==0 ? 1 : 0.4);
  });

});
</script>

</head>

<body>
<h1>Select an electronic ID:</h1>

<div id="content">
    <div onClick="logout()">[Logout]</div>
</div>

<div id="box_myid" class="example">

    <div class="demo">

    <div onClick="myID_click();" class="selectorClass demoObjectMyID"></div>

  </div>
    <div class="clear"></div>
</div>


<div class="example" id="box_bankid">

    <div class="demo">

    <div class="selectorClass demoObject"></div>

  </div>
    <div class="clear"></div>
```

```
</div>


<div class="example" id="box_buypass">

    <div class="demo">

    <div class="selectorClass demoObjectBuyPass"></div>

  </div><div class="clear"></div>
</div>



<div class="example" id="box_minid">

    <div class="demo">

    <div class="selectorClass demoObjectMinID"></div>

    </div>
    <div class="clear"></div>
</div>
<div id="cookiebutton">

</div>
</body>
</html>
```

Listing C.2: landing.php on RP

```php
<?php

session_start();

function __autoload($f) {
    //load the interfaces first otherwise contract errors occur
    $interfaceFile = "classes/interface/" . $f . "Interface.php";

    if (file_exists($interfaceFile)) {
        require_once $interfaceFile;
    }

    //load class files after interfaces
    $classFile = "classes/" . $f . ".php";
    if (file_exists($classFile)) {
        require_once $classFile;
    }

    //if utilities are needed load them last
    $utilFile = "classes/util/" . $f . ".php";
    if (file_exists($utilFile)) {
        require_once $utilFile;
    }
}


if(extension_loaded('gmp') && !defined('USE_EXT')){
    define ('USE_EXT', 'GMP');
}else if(extension_loaded('bcmath') && !defined('USE_EXT')){
    define ('USE_EXT', 'BCMATH');
}




$Qx = '6174796913462593303486602434131194488337606921407879712879';
$Qy = '4571935574014424850307214357578070306631117006599250439368';
$G = NISTcurve::generator_192();
```

```
$curve_192 = NISTcurve::curve_192();
$Q = new Point($curve_192, $Qx, $Qy);



function verify_signature($Rx, $Ry, $s, $ch) {
GLOBAL $curve_192, $Q, $G;

$H1 = hash_1($ch);
$h2_input = '('. $Rx . ',' . $Ry. ')';
$h2 = bcmath_Utils::bchexdec(hash('sha256', $h2_input, false));
$R = new Point($curve_192, $Rx, $Ry);
$left =  Point::add($H1, Point::mul($s, $G));
$right = Point::add($R, Point::mul($h2,$Q));
if ($left->x == $right->x && $left->y == $right->y) {return true;}
else return false;
}



function hash_1($ch){
GLOBAL $G, $curve_192;

$a = $curve_192->getA();
$b = $curve_192->getB();
$p = $curve_192->getPrime();
$hash = bcmath_Utils::bchexdec(hash('sha256', $ch, false));
$t = '1569275433846670190958947355801916604020977175097581240320';
$exp = bcdiv(bcsub($p,'1'),'2');
$pre_z = bcadd(bcadd(bcpow($hash,"3"), bcmul($a,$hash)),$b);
$z = bcmod($pre_z, $p);
$remainder = bcpowmod($z, $exp, $p);

while ($remainder != 1) {
$hash = bcadd($hash,'1');
$pre_z = bcadd(bcadd(bcpow($hash,"3"), bcmul($a,$hash)),$b);
$z = bcmod($pre_z, $p);
$remainder = bcpowmod($z, $exp, $p);
}

$y = bcpowmod($z, $t, $p);
```

```php
return new Point($curve_192, bcmod($hash,$p), $y);

}


if (!isset($_SESSION['ch']))
{
 die('No challenge detected for current session');
}

if (!isset($_GET['Rx']) || !isset($_GET['Ry']) || !isset($_GET['s']))
{
 die('missing signature');
}



$sigver = verify_signature($_GET['Rx'], $_GET['Ry'], $_GET['s'], $_SESSION['ch']);
if ($sigver === true) {
$_SESSION['auth'] = "true";
header("Location: index.php");}
else die('signature verification failed!');

?>
```