Partially blind password-based signatures using elliptic curves

Kristian Gjøsteen

July 31, 2013

Abstract

Password-based signatures allow a user who can only remember a password to create digital signatures with the help of a server, without revealing the messages to be signed to the server.

Certain applications require the ability to disclose part of the message to the server. We define partially blind password-based signatures and construct a scheme based that we prove secure, based on a novel computational problem related to computing discrete logarithms.

Our scheme is based on Nyberg-Rueppel signatures. We give a variant of Nyberg-Rueppel signatures that we prove secure based on our novel computational problem.

Unlike previous password-based signature schemes, our scheme can be instantiated using elliptic curve arithmetic over small prime fields. This is important for many applications.

1 Introduction

Digital signature schemes are useful, both as primitives for use in cryptographic protocols and directly for creating legally meaningful electronic signatures. A digital signature scheme poses two key management problems, that of distributing the (public) verification key and keeping the (private) signing key both available and secret.

Modern computers are generally considered insecure, and if a signing key is stored on a compromised computer, the key is vulnerable to theft. The traditional approach to prevent theft is to store the signing key on a smart card (or equivalent hardened device) that the owner of the key controls physically. The smart card is connected to a computer and signs the messages the computer gives it. Note that unless special care is taken, a smart card only protects against key theft, not key misuse by a compromised computer.

Smart cards have never been easy to deploy. Today, many modern computing devices such as smart phones and tablets cannot be connected to smart cards. This means that smart cards are becoming even more difficult to deploy.

Many applications do not require high security levels, and perhaps user-chosen passwords and one-time codes sent out-of-band (e.g. SMS messages) are considered sufficient. A low baseline for security may also be sufficient as long as motivated users can do extra (maybe one-time) work to increase their security. It therefore makes sense to ask if alternative solutions to smart cards are possible.

One possible solution is to have users authenticate to a central server which then signs messages on behalf of the users. Users are comfortable with passwords and a careful implementation could provide reasonable security [6]. However, there are obvious privacy issues, and one may also consider the risk of a corrupt server impersonating users. The risk of corrupt servers could be mitigated by sharing the signing key [9] between independently run servers or using tamper-resistant hardware [11], but finding convincingly independent parties is often difficult.

Gjøsteen and Thuen [7] provide a primitive that can be used to reduce the trust requirements in such schemes, namely password-based signatures. The idea is that the signing operation should depend on a user-chosen password in such a way that signatures cannot be created without the user's password. The user can then protect against a corrupt server by choosing a good password.

The naïve approach is to treat the password as a seed to a pseudo-random generator which is then used to generate a key for a traditional signature scheme. This approach is too simple. The system must provide some security even when passwords are chosen according to low-entropy probability distributions. Because guesses can be checked against the public key, exhaustive search will work.

Password-based signatures must rely on a server to help generate signatures. The general strategy is that unlike the trusted third party discussed above, the server does not have the entire signing key, just a share of it. The user's password is the second share, and generating a signature is done using a two-party protocol.

There are three natural security goals for a password-based signature scheme:

- it should be impossible to generate a signature without the server's participation, so that any authentication systems restricting access to the server will be effective;
- it should be impossible for the server to generate a signature without first guessing the user's password, so that users can protect themselves against corrupt servers by choosing good passwords; and
- it should be hard for the server to say anything about which messages the user signs, to protect the user's privacy.

Application requirements presents us with two challenges. First of all, we want to be able to deploy such a scheme to a variety of devices. This is possible if we can execute the required protocols using Javascript code run inside modern web browsers.

Unfortunately, it is difficult to implement fast large integer arithmetic in Javascript, which is a problem for modern cryptography. (Future language developments along the lines of Webcrypto could change this, but reliable implementations are not expected soon.) However, it is possible to use cryptography based on ordinary elliptic curves over small prime fields in Javascript [8]. Compared to native code, point multiplications are expensive, but feasible.

Gjøsteen and Thuen [7] provided two schemes, one based on bilinear pairings and one based on RSA. Unfortunately, the bilinear scheme is too computationally intensive to be practical in our setting. The RSA scheme's signing algorithm might be feasible, but joint RSA key generation is required, which is too computationally expensive and not feasible.

A second challenge is that in a trade-off between privacy and security, certain applications [6] need to disclose part of the message to be signed to the server. Password-based signatures do not satisfy this application requirement.

Our Contribution In this work we extend the notion of password-based signatures to *partially blind* password-based signatures, allowing the user to disclose part of the message to be signed to the server. We design and analyse a partially blind password-based signature that can be based on ordinary elliptic curves over small prime fields.

Our scheme is based on Nyberg-Rueppel signatures [12]. Signature verification costs two point multiplications. The key generation protocol requires one point multiplication by both parties. Generating a signature again costs the server one point multiplication and the user three point multiplications (since the user must verify the signature before accepting).

We have not been able to find a proof of security based on standard computational assumptions, but we have identified a problem related to computing discrete logarithms that allows security proofs. Based on this problem, we are also able to prove the security of a modified Nyberg-Rueppel signature scheme in the random oracle model.

We provide some evidence for the hardness of this problem by proving that it is hard in the generic group model. Obviously, this does not guarantee that the problem is hard in the real world for any reasonable parameter size, but given the similarity with the discrete logarithm problem, it seems reasonable to assume its hardness for common parameter sizes.

We note that an additional benefit of identifying such problems is to identify exactly what effects of the random oracle model we rely on, and at the same time how the cryptosystem could be attacked by mathematical analysis.

Related Work Password-based signatures are to a certain extent inspired by blind signatures, first introduced by Chaum [4], and later formalized by Juels, Luby and Ostrovsky [10]. Partially blind signatures were proposed by Abe and Fujisaki [1].

Variants of Nyberg-Rueppel signatures have been analysed previously [2] directly in the generic group model. In contrast, we identify a computational problem and only use the generic group model heuristic to analyse the computational problem. The problem with the direct approach is that we know there are cryptographic schemes that can be proven secure in the generic group model, yet cannot be instantiated securely. In other words, the generic group model heuristic is far from perfect. We believe that the generic group model is a much better heuristic analysis tool when applied to computational problems than when applied to cryptosystems.

Blind signatures based on Nyberg-Rueppel signatures were proposed by Camenisch, Piveteau and Stadler [3]. As can be seen, our construction is quite different. We use hash functions extensively, and we have a two-move protocol, while the earlier blind signature scheme is a three-move protocol. This is due both to the improved understanding of security requirements, and to the fact that security requirements for password-based signatures can be simpler than security requirements for blind signatures.

Many other approaches to useable digital signatures have appeared in the literature. For instance, Damgård and Mikkelsen [5] consider a scenario where a user has access to second (less capable) terminal that communicates with the main terminal. The main practical problem is that communication between terminals using wireless technologies is surprisingly difficult, even today.

As mentioned above one could probably also approach the problem by using multiple signing servers working together [9]. One downside of this approach is that it requires a more expensive infrastructure and several rounds of computation.

Overview Section 2 augments the existing definition [7] of a password-based signature scheme to include partial blindness and discusses the relevant security goals. We also improve slightly on the existing notion of unframeability.

Section 3 defines our novel problem and analyses it in the generic group model. Section 4 uses this problem to provide a novel analysis of a modified Nyberg-Rueppel signature scheme.

Section 5 defines our new partially blind password-based signature scheme and proves it secure under the problem defined in Section 3. We note that the proof strategy used is very similar to the one used to prove the modified Nyberg-Rueppel scheme secure in Section 4.

Section 6 raises certain issues to be aware of when using the password-based signature primitive to design larger systems. It also outlines some issues for future work.

2 Partially Blind Password-based Signatures

The following definitions are based on those in Gjøsteen and Thuen [7]. The functional definitions have been extended to include partial message disclosure. We have also slightly improved the definition of non-frameability.

We consider the case where a *user* wants to sign something, but since he can only store a *password*, not a signing key, he needs the assistance of a *server* to sign.

Formally, a password-based signature scheme consists of a *verification* algorithm and two two-party protocols between a user $\mathcal{U}$ and a server $\mathcal{S}$, one for *key generation* and one for *signing*. There may also be an additional *setup* algorithm, run by a trusted third party, perhaps for choosing common parameters or sampling a common reference string.

In the key generation protocol, $\mathcal{U}$ gets a secret password as input. When the protocol ends, $\mathcal{U}$ outputs a *verification key* and $\mathcal{S}$ outputs a *verification key* and a *signing key share*.

In the signing protocol, $\mathcal{U}$ gets a verification key, a two-part message and a secret password as input, and $\mathcal{S}$ gets the verification key, the second part of the message and a signing key share as input. When the protocol ends, $\mathcal{U}$ outputs a signature on the two-part message, or $\perp$.

The verification algorithm takes as input a verification key, a two-part message and a signature, and outputs either 1 or 0.

The completeness requirement is that, when the verification key and the signing key share were generated by the key generation protocol, and the signature was generated by the signing protocol, the verification algorithm will output 1.

We shall restrict our attention to two-move signing protocols following the traditional pattern request–issue–unblind. A *request* algorithm $\mathcal{R}$ gets as input a verification key, a two-part message and a password and outputs a request and a state. An *issue* algorithm $\mathcal{I}$ gets as input a message part, a verification key, a signing key share and a request, and outputs a blinded signature. An *unblind* algorithm $\mathcal{UB}$ gets as input a state and a blinded signature, and outputs a signature.

The obvious protocol runs as follows: $\mathcal{U}$ runs $\mathcal{R}$ and sends the request to $\mathcal{S}$, which runs $\mathcal{I}$ on the request. Then $\mathcal{S}$ sends the blinded signature to $\mathcal{U}$, which in turn runs $\mathcal{UB}$ to get the final signature. The completeness requirement for the signing protocol translates into natural completeness requirements for the three algorithms.

2.1 Security Notions

There are three natural security notions for password-based signatures. *Restricted blindness* says that the server should not learn anything about the part of the message it does not get as input. *Non-forgeability* says that a user should not be able to sign a message without access to the server. And *non-frameability* says that server cannot forge signatures without first guessing the user's password. (The assumption is that it will always be much easier to guess the user's password than to break the underlying mathematical problems.)

Passwords are bit strings sampled from some (possibly low-min-entropy) probability space. We say that a probability space $\mathcal{PW}$ is f*-cumulative* if any algorithm using at most T test queries will guess a password sampled from $\mathcal{PW}$ with probability at most $f(T)$.

We restrict attention to two-move signing protocols as discussed above, since this significantly simplifies both definitions and analysis.

We also limit ourselves to considering restricted blindness. Gjøsteen and Thuen [7] define stronger notions of blindness where the adversary is allowed to see the signatures generated. For blind signatures, where the idea is that there are many users per signing key, these stronger blindness notions are very important for applications.

For password-based signatures, however, where there is only one user, these notions are less useful. Once the adversary sees the signature on the message, all that remains to be discovered is exactly when the signature was generated. We expect that most applications will include some form of time stamp in the signatures, rendering this issue moot. However, it is conceivable that there are applications that require stronger forms of blindness. Our scheme will not be suitable for such applications.

Restricted Blindness *A corrupt server running the signing protocol should not be able to say anything about the message included in a request, based on the request only.*

We consider the following game between an adversary and a simulator. First, the adversary gives the simulator a verification key and a password. The simulator flips a coin b. Then the adversary sequentially gives the simulator pairs of messages $(m_1^{(0)}, m_2), (m_1^{(1)}, m_2)$. The simulator replies to each pair with requests for $(m_1^{(b)}, m_2)$ and $(m^{(1-b)}, m_2)$.

The adversary wins if in the end he outputs a bit b' that equals the simulator's bit b.

The adversary's advantage is the distance between the probability that he wins and $1/2$. We say that the system is (T, n, ϵ)*-restricted-blind* if any adversary using time at most T and making at most n signing requests has advantage at most ϵ.

Non-forgeability *A corrupt user running the signing protocol n times should not be able to produce $n + 1$ valid signatures.*

We consider the following game between an adversary and a simulator. First, the key generation protocol is run with the adversary playing the user role and the simulator playing the server. Then the adversary runs and may make n requests to the simulator, upon which the simulator runs the issue algorithm $\mathcal{I}$ with appropriate input and returns the blinded signatures.

The adversary wins if in the end he outputs $n + 1$ valid signatures on $n + 1$ distinct messages.

The adversary's advantage is the probability that he wins. We say that the system is (T, n, ϵ)*-non-forgeable* if any adversary using time at most T and making at most n signing requests has advantage at most ϵ.

Non-frameability *A corrupt server should not be able to sign without first guessing the user's password.*

We consider the following game between an adversary and a simulator, conditional on a f-cumulative probability space $\mathcal{PW}$. First, the simulator samples a password from $\mathcal{PW}$ and runs the key generation protocol with the adversary as the server. The adversary then asks the simulator to make up to n signature requests. After the adversary replies to these requests, the simulator unblinds the signatures and passes them on to the adversary.

The adversary wins if he eventually outputs a signature on a message that the user has not made a request for.

The adversary's advantage is the probability that he wins this game. We say that a system is (T, n, ϵ)-*non-frameable* if for any f-cumulative password space $\mathcal{PW}$ any adversary using time at most T has advantage at most $\epsilon + f(T)$.

Summary A password-based signature scheme is (T, n, ϵ)-(restricted-)secure if it is (T, n, ϵ)-(restricted-)blind, -non-forgeable and -non-frameable.

3 The Underlying Problem

We shall now define a novel computational problem that is sufficient to prove a modified version of Nyberg-Rueppel signatures and our scheme secure. The problem is essentially to find the discrete logarithm of the product of a freely chosen group element and a random power of a challenge element.

The purpose of defining this problem is to identify exactly what effects of the random oracle model we rely on, and which group properties are needed.

Let G be a cyclic group of order q. Let g be a generator. Even though the intention is that G is the rational points on an elliptic curve over a finite field under the usual group law, we use multiplicative notation for G, since that makes our equations easier to read.

Problem 1. *Suppose y is a group element chosen uniformly at random. We have access to an oracle that*

- *on input of a* challenge *r (which may have been queried before), chooses a random integer t, $0 \leq t < q$, records (r, t) and replies with t.*

The problem is to find an s such that the relation

$$g^s = ry^t$$

holds for some pair (r, t) recorded by the oracle.

We say that an adversary is a (T, ϵ)-adversary against this problem if the adversary uses time at most T, makes at most T queries to the oracle and outputs a valid relation with probability at least ϵ.

It is clear that this problem is no more difficult than computing discrete logarithms. It is certainly also plausible that it is no easier than computing discrete logarithms, but we do not believe it is possible to prove this.

Generic Model Analysis We consider Problem 1 in the generic group model. This does not prove that the problem will be hard for a specific group, but at least we know that the problem is not obviously easy.

A generic group algorithm is an algorithm that does not care about the representation of group elements. Formally, we consider a randomly chosen bijection $\sigma : \{0, 1, 2, \dots, q-1\} \to \mathbb{Z}_q$. The generic group algorithm does not have direct access to σ, but may query an oracle with

- a single element z, to which the oracle reply is $\sigma^{-1}(-\sigma(z))$; or
- a pair of elements (z, z'), to which the oracle reply is $\sigma^{-1}(\sigma(z) + \sigma(z'))$.

The generic algorithm is typically given $\sigma^{-1}(1)$ as input.

An alternative oracle is one that maintains a partial injection $\sigma : \{0, 1, 2, \dots, q-1\} \to \mathbb{Z}_q$. When a query involves an element z not in the domain of σ, or the result u is not in the image of σ, it chooses u or z at random from the the elements not in the image or domain of σ, and extends σ with $z \mapsto u$. This oracle is more convenient for proofs.

Theorem 1. *Any generic group algorithm making at most n queries to the group oracle and challenge oracle has probability at most $n(n+1)/2q$ of outputting a valid triple.*

Proof. We phrase the proof as a sequence of games, and begin with the game where the generic group algorithm interacts with the generic group oracle as well as the challenge oracle. We let $\sigma(y) = a$.

We modify this game by switching to the alternative generic group oracle, as well as adding a second representation of group elements, $\rho : \{0, 1, 2, \dots, q-1\} \to \mathbb{Z}_q[X]$, where $\rho(g) = 1$ and $\rho(y) = X$.

It can happen at some point that we compute a value $u + vX \in \mathbb{Z}_q[X]$ that is not in the image of ρ, but the corresponding value $u + va$ is in the image of σ, say $\sigma(z) = u + va$. If this happens, we stop the game, which is an exceptional event.

In this case, $\rho(z) = u' + v'X$, which means that a is a zero of the polynomial $u - u' + (v - v')X$. Until such a collision happens, the generic algorithm only knows what the value of a cannot be. After n queries without a collision, there are at most $n(n-1)/2$ linear polynomials that a cannot be be a zero of. We conclude that the probability of the exceptional event is upper-bounded by $n(n-1)/2q$.

Suppose the generic algorithm does output a triple (r, s, t) that the simulator accepts, and that $\rho(r) = u + vX$. Since ρ is injective, we get that

$$s \equiv u + vX + tX \pmod{q}.$$

It follows that $v \equiv -t \pmod{q}$.

Since r was queried in a challenge query before t was sampled, we know that v was fixed before t was chosen at random. Therefore, the probability that $v \equiv -t \pmod{q}$ is $1/q$. Since there are at most n challenge queries, the probability of this happening is upperbounded by n/q.

It now follows that the adversary's probability of success in this game is at most n/q, from which the claim follows. $\square$

4 Modified Nyberg-Rueppel Signatures

The Nyberg-Rueppel signature scheme [12] is similar to ElGamal and DSA. Consider the following modified Nyberg-Rueppel signature scheme, using two hash functions $H_1 : \{0,1\}^* \to G$ and $H_2 : G \to \{0,1,\dots,q-1\}$. We note that for elliptic curves, it is easy to construct efficient hash functions from bit strings into the group.

Key generation chooses a random number a from $\{0,1,2,\dots,q-1\}$ as the signing key, and computes $y = g^a$ as the verification key.

To *verify* a signature (r,s) on a message m using verification key y, we check the verification equation

$$H_1(m)g^s \overset{?}{=} ry^{H_2(r)}.$$

We use this specific verification equations because it slightly simplifies the presentation. Other equivalent equations could improve computational efficiency.

To *sign* a message m using signing key a, choose a number k at random from $\{0,1,2,\dots,q-1\}$, compute $r = H_1(m)g^k$ and $s = k + H_2(r)a$. The signature is (r,s).

A (T,n,ϵ)-*adversary* against the signature scheme is an adversary using at most time T, making at most n queries to a signature oracle, that has probability ϵ in producing a valid forgery.

The security proof begins with the standard signature game. The main change is to the signing oracle so that instead of using the signing key, it tampers with the H_2 hash function. The only complication is that depending on the adversary's successful forgery, there are two possibilities for extracting the required relation, but which to use must be decided before the adversary's produces his forgery. This accounts for the halving of the advantage.

Theorem 2. *Given a (T,n,ϵ)-adversary against the signature scheme, we can construct a $(T,\epsilon/2 - T^2/q)$-adversary against Problem 1.*

Proof. We begin with a game between the signature game simulator and the adversary. First we subsume the random oracles inside the simulator.

Next, we change the signing oracle. Note that the adversary interacting with the original signing oracle expects s to be uniformly distributed. Instead of choosing r and computing s, the simulator chooses random s and t, computes r using the formula

$$r = H_1(m)g^s y^{-t}$$

and programs the H_2 oracle to hash r to t. The only way this can fail is if the programming of the hash oracle fails, that is, if the H_2 oracle has already been queried or programmed at r. Since r is chosen uniformly at random, this happens with probability at most $nT/q \le T^2/q$.

If the simulator ever encounters a collision in H_1, it stops. The probability that this happens is upperbounded by T^2/q.

Next, we change the H_1 oracle, so that when it is asked to hash a message m, it does not just choose a random group element as the hash value, but instead chooses a random number u and uses either y^u or g^u as the hash value.

If the adversary interacting with the original simulator succeeds with probability ϵ, an adversary interacting with our modified simulator succeeds with probability at least

$$\epsilon' = \epsilon - (T^2/q + T^2/q) = \epsilon - 2T^2/q.$$

We now see that we have a simulator that chooses y uniformly at random at the start, that upon queries for r responds with a random integer t. The only remaining issue is how to convert the adversary's signature to the required (r, s, t)-relation.

An adversary interacting with our modified simulator that successfully creates a valid forgery (r, s) on a message m, will either use an r that was first queried to the H_2 oracle, or an r from a signature returned by the signing oracle.

In the former case, if messages have been hashed to known (to the simulator) powers of g, the verification equation for the valid forgery then becomes

$$ry^{H_2(r)} = H_1(m)g^s = g^u g^s.$$

In other words, we have the required relation.

In the latter case, there was a reply (r, s') on a signing request for the message m' (which cannot equal m, since then $s = s'$). In this case, if m and m' have been hashed to y^u and $y^{u'}$, respectively, we have that

$$H_1(m)g^s = ry^{H_2(r)} \qquad \text{and} \qquad H_1(m')g^{s'} = ry^{H_2(r)}.$$

Combining the two equations, we get

$$H_1(m)g^s = H_1(m')g^{s'},$$

or

$$y^{u-u'} = g^{s'-s}.$$

Since there are no collisions in H_1, we can recover $\log_g y$, which allows us to easily generate a relation of the required form.

Depending on how we construct the hashes of messages, we have two adversaries for our problem. Choosing between them at run-time with probability 1/2 yields a combined adversary with the claimed advantage. □

5 Our Construction

Our construction is based on a slightly modified Nyberg-Rueppel verification equation, where we add randomness before hashing the message. To accomodate the revealed part of the message, the domain of the hash function H_2 must be $G \times \{0,1\}^*$, not just G.

The *key generation protocol* is quite simple, and requires a hash function $H_4 : G \to \{0,1\}^l$, where we shall assume that $2^l < q$. The user computes its part of the verification key $y_{\mathcal{U}} = g^{H_3(pw)}$. The server chooses its signing key share a uniformly at random from $\{0,1,2,\ldots,q-1\}$ and computes its part of the verification key $y_{\mathcal{S}} = g^a$. Both parties commit to their part of the verification key by sending $H_4(y_{\mathcal{U}})$ / $H_4(y_{\mathcal{S}})$ to the other party. Afterwards, both parties open the commitments by sending $y_{\mathcal{U}}$ / $y_{\mathcal{S}}$ to the other party. The verification key is $y = y_{\mathcal{U}} y_{\mathcal{S}}$.

To *verify* a signature (k_0, r, s) on a two-part message (m_1, m_2) using the verification key y, we check the verification equation

$$H_1(m_1 || k_0) g^s \stackrel{?}{=} r y^{H_2(r, m_2)}.$$

The *request algorithm* $\mathcal{R}$ chooses a random string k_0 of length l and a random integer $k_1 \in \{0,1,2,\ldots,q-1\}$. It computes the *request* $r_0 = H_1(m_1||k_0)g^{k_1}$. The *state* is y, m_1, m_2, k_0, k_1 and pw.

The *issue algorithm* chooses a random integer $k_2 \in \{0,1,2,\ldots,q-1\}$ and computes $r = r_0 g^{k_2}$ and $s_0 = H_2(r, m_2)a + k_2 \bmod q$ and outputs the *blinded signature* (r, s_0).

The *unblind algorithm* computes $s = s_0 + H_2(r, m_2)H_3(pw) + k_1 \bmod q$, verifies that the verification equation holds, and outputs the signature (k_0, r, s).

The system is *complete*, since

$$H_1(m_1||k_0)g^s = H_1(m_2||k_0)g^{k_1+k_2} g^{(a+H_3(pw))H_2(r,m_2)} = r y^{H_2(r)}.$$

5.1 Security Analysis

The security claim for our system is captured in the following theorem. It follows from the Propositions 1–3 below, proving blindness, non-forgeability and non-frameability.

Theorem 3. *Given a (T, n, ϵ)-restricted-adversary against the scheme, we can construct a $(T, \epsilon/2 - 5T^2/2^{l+1})$-adversary against Problem 1.*

Restricted blindness Restricted blindness is much easier than than full blindness, since we do not have to worry about the complete signature, only the request. For our scheme, we observe that the request is statistically independent of the message, and blindness is immediate. We also note that our scheme does not achieve blindness.

Proposition 1. *The scheme is $(T, n, 0)$-restricted-blind.*

Proof. A request r_0 is distributed uniformly at random and is independent of the message. Any adversary that sees any number of requests will therefore be correct exactly half the time and the claim follows. □

Non-forgeability The proof of non-forgeability is very similar to the security proof for the modified Nyberg-Rueppel scheme. The main new idea is that we tamper with the key generation protocol to force a suitable value for the verification key, which means that we can derive the appropriate relation from the list of signatures the adversary generates. As before, we issue signatures by tampering with the H_2 oracle.

Proposition 2. *Given a (T, n, ϵ)-adversary against non-forgeability, we can construct a $(T, n, \epsilon/2 - 5T^2/2^{l+1})$-adversary against Problem 1.*

Proof. We begin with a game between the non-forgeability simulator and the adversary. First we subsume the random oracles inside the simulator.

Next, we change the signing oracle. Note that the adversary interacting with the original issue algorithm expects s_0 to be randomly distributed and that $ry^{H_2(r,m_2)} = r_0 g^{s_0} y_{\mathcal{U}}^{H_2(r,m_2)}$. When our simulator receives a request r_0, it chooses random s_0 and t, computes r using the equation

$$r = r_0 y^{-t} g^{s_0} y_{\mathcal{U}}^{t},$$

and programs the H_2 oracle to hash (r, m_2) to t. The only way this can fail is if the programming of the hash oracle fails, that is, if the oracle has already been queried at (r, m_2). Since r is chosen at random, the probability that this happens is upperbounded by Tn/q.

If the simulator ever encounters a collision in H_1 or H_2, it stops. The probability of this happening is upperbounded by T^2/q.

Next, we change the key generation protocol as follows. When the simulator receives the adversary's commitment, it checks if the commitment has exactly one preimage. If it has no preimages or more than one, we stop. The probability that the latter happens is upperbounded by $T^2/2^l$, where l is the length of the commitments. The probability that the adversary will be able to complete the protocol if there is no preimage is upperbounded by $T/2^l$.

The next change is that the simulator chooses y at random. Instead of committing to $y_{\mathcal{S}}$, it sends a random hash value. After the simulator has deduced $y_{\mathcal{U}}$, it computes $y_{\mathcal{S}} = y/y_{\mathcal{U}}$ and programs the H_4 oracle accordingly. This programming fails only if H_4 has been queried at $y_{\mathcal{S}}$ before, the probability of which can be upperbounded by T/q.

Finally, we change the H_1 oracle, so that when it is asked to hash a message $m_1||k_0$, it does not just choose a random group element as the hash value, but instead either chooses an exponent u and uses g^u as the hash value, or chooses two exponents u and v and uses $g^u y^v$ as the hash value.

If the adversary interacting with the original simulator succeeds with probability ϵ, an adversary interacting with our modified simulator succeeds with probability at least

$$\epsilon' = \epsilon - (Tn/q + T^2/q + T^2/2^l + T/2^l + T/q) \geq \epsilon + 1/q - 5T^2/2^l.$$

We now see that we have a simulator that chooses y uniformly at random at the start, that upon queries for r (and m_2) responds with a random integer

t, and at most n times receives a request r_0 for which it finds (r, s, t), but t was chosen before r. The only remaining issue is how to convert the adversary's signatures to the required (r, s, t)-relation.

An adversary interacting with our modified simulator that creates $n+1$ valid signatures on $n+1$ distinct messages, will either use $n+1$ distinct r's, or produce signatures on two distinct messages that have the same r.

In the former case, if messages have been hashed to known (to the simulator) powers of g, we get from each signature a distinct relation

$$g^{s+u} = ry^{H_2(r,m_2)}.$$

At most n of those relations will involve an r for which our simulator chose t before it selected r. For at least one relation, t must have been chosen randomly in response to r. In other words, we have the required relation.

In the latter case, we have two distinct message pairs (m_1, m_2) and (m_1', m_2') with signatures (k_0, r, s) and (k_0', r, s'). If $m_1||k_0 = m_1'||k_0'$, then $m_2 \neq m_2'$ and the verification equations for the two signatures give us

$$g^{s-s'} = y^{H_2(r,m_2)-H_2(r,m_2')},$$

Since there are no collisions in H_2, we can recover $\log_g y$, which allows us to easily generate a relation of the required form.

Assume that $m_1||k_0 \neq m_1'||k_0'$. If $m_1||k_0$ and $m_1'||k_0'$ have been hashed to $g^u y^v$ and $g^{u'} y^{v'}$, respectively, we get that

$$\begin{aligned} H_1(m_1||k_0)g^s = g^{s+u}y^v &= ry^{H_2(r,m_2)} = ry^{H_2(r,m_2')}y^{H_2(r,m_2)-H_2(r,m_2')} \\ &= H_1(m_1'||k_0')g^{s'}y^{H_2(r,m_2)-H_2(r,m_2')} \\ &= g^{s'+u'}y^{v'}y^{H_2(r,m_2)-H_2(r,m_2')}. \end{aligned}$$

There are no collisions in H_1, so we know that the pairs (u, v) and (u', v') are distinct. Since v and v' are chosen independently and uniformly at random, we may consider the difference $v - v' \bmod q$ to be chosen uniformly at random.

The adversary has no information about v and v'. Therefore, the difference $H_2(r, m_2) - H_2(r, m_2') \bmod q$ (which to a certain extent may be chosen by the adversary) is independent of the difference $v - v' \bmod q$. It follows that we may consider

$$v - v' - (H_2(r, m_2) - H_2(r, m_2')) \bmod q$$

to be chosen uniformly at random, which means that it will be zero with probability $1/q$. When this value is non-zero, we can recover $\log_g y$, which allows us to easily generate a relation of the required form.

Depending on how we construct the hashes of messages, we have two adversaries for our problem. Choosing between them at run-time with probability $1/2$ yields a combined adversary with the claimed advantage. □

Non-frameability The proof of non-frameability is very similar to the proof of non-forgeability. Instead of tampering with the H_2 oracle, we now use the randomness included with the message in H_1 to fake signing messages. Note that the adversary now produces only one signature.

Proposition 3. *Given a (T, n, ϵ)-adversary against non-frameability, then we can construct a $(T, n, \epsilon/2 - 5T/2^{l+1})$-adversary against Problem 1.*

Proof. We begin with a game between the non-frameability simulator and the adversary, with the random oracles subsumed within the simulator. Let $\mathcal{PW}$ be an f-cumulative password space, and suppose our adversary has advantage $\epsilon + f(T)$.

Next, if the adversary queries H_3 at pw, we stop. By assumption, this happens with probability at most $f(T)$, leaving the adversary with a success probability of at least ϵ.

We modify the request and unblind process. Note that since H_1 is a random function queried at a random point, the adversary will almost never have any information about k_1, so the final s will appear to be uncorrelated to s_0. Instead of issuing a request and unblinding the result, our modified simulator does the following: The request is computed as $r_0 = g^{k_1}$. To unblind after receiving (r, s_0), we first use the alternative verification equation

$$r_0 g^{s_0} \overset{?}{=} r y_{\mathcal{S}}^{H_2(r,m_2)}.$$

Then we choose random k_0 and s, compute $h = g^{-s} r y^{H_2(r,m_2)}$, and program the H_1 oracle to hash $m_1||k_0$ to h. The only way this can fail is if the programming of the hash oracle fails, that is, if the H_1 oracle has already been queried at $m_1||k_0$. This happens with probability at most $T/2^l$.

If the simulator encounters a collision in H_1 or H_2, it stops. The probability of this happening is upperbounded by T^2/q.

Next, we change the key generation protocol as in the proof of Proposition 2, except that the simulator plays the role of the user, not the server. The end result is that the simulator samples y uniformly at random, and computes $y_{\mathcal{U}} = y/y_{\mathcal{S}}$. This fails with probability at most $T^2/2^l + T/2^l + T/q$.

Finally, we change the H_1 oracle, so that when it is asked to hash a message $m_1||k_0$, it does not just choose a random group element as the hash values, but instead either chooses an exponent u and uses g^u as the hash value, or chooses two exponents u and v and uses $g^u y^v$ as the hash value.

If the adversary interacting with the original simulator succeeds with probability $\epsilon + f(T)$, an adversary interacting with our modified simulator succeeds with probability at least

$$\begin{aligned}\epsilon' &= \epsilon + f(T) - (f(T) + T/q + T^2/q + T^2/2^l + T/2^l + T/q) \\ &\geq \epsilon + 1/q - 5T^2/2^l.\end{aligned}$$

We now see that we have a simulator that chooses y uniformly at random at the start, that upon queries for r responds with a random integer t. The only remaining issue is how to convert the adversary's signature to the required (r, s, t)-relation.

There are two cases to consider. When the adversary outputs a valid signature (k_0, r, s) on a message pair (m_1, m_2), then either the adversary's r comes from a signature created by the simulator, or it does not.

In the latter case, if H_1 hashed $m_1||k_0$ to g^u, we get that

$$ry^{H_2(r,m_2)} = H_1(m_1||k_0)g^s = g^{u+s}.$$

In other words, we have the required relation.

In the former case, we have two distinct message pairs $(m_1, m_2), (m_1', m_2')$ and two signatures (k_0, r, s) and (k_0', r, s'). If $m_1||k_0 = m_1'||k_0'$, then $m_2 \neq m_2'$ and the verification equations for the two signatures give us

$$g^{s-s'} = y^{H_2(r,m_2)-H_2(r,m_2')}.$$

Since there are no collisions in H_2, we can recover $\log_g y$, which allows us to easily generate a relation of the required form.

Assume that $m_1||k_0 \neq m_1'||k_0$. If $m_1||k_0$ and $m_1'||k_0$ have been hashed to $g^u y^v$ and $g^{u'} y^{v'}$, respectively, we get that

$$\begin{aligned} H_1(m_1||k_0)g^s &= g^u y^v g^s = ry^{H_2(r,m_2)} = ry^{H_2(r,m_2')} y^{H_2(r,m_2)-H_2(r,m_2')} \\ &= H_1(m'||k_0')g^{s'} y^{H_2(r,m_2)-H_2(r,m_2')} \\ &= g^{u'} y^{v'} g^{s'} y^{H_2(r,m_2)-H_2(r,m_2')}. \end{aligned}$$

There are no collisions in H_1, so we know that the pairs (u, v) and (u', v') are distinct. Since v and v' are chosen independently and uniformly at random, we may consider the difference $v - v' \bmod q$ to be chosen uniformly at random.

The adversary has no information about v and v'. Therefore, the difference $H_2(r, m_2) - H_2(r, m_2') \bmod q$ (which to a certain extent may be chosen by the adversary) is independent of the difference $v - v' \bmod q$. It follows that we may consider

$$v - v' - (H_2(r, m_2) - H_2(r, m_2')) \bmod q$$

to be chosen uniformly at random, which means that it will be zero with probability $1/q$. When this value is non-zero, we can recover $\log_g y$, which allows us to easily generate a relation of the required form.

Depending on how we construct the hashes of messages, we have two adversaries for our problem. Choosing between them at run-time with probability 1/2 yields a combined adversary with the claimed advantage. $\square$

6 Concluding Remarks

We have expanded on the definition of password-based signatures to include the possibility of partial message disclosure, which may be required by applications. We have defined a new partially blind password-based signature scheme that can be based on elliptic curves, and proved it secure under a novel hardness assumption. We have proven that this hardness assumption holds in the generic model. We have also defined a variant of the Nyberg-Rueppel signatures scheme and proved it secure under the same hardness assumption.

Changing passwords must be possible. Since the key generation protocol used in our scheme is very cheap, the obvious approach is to recall the user's previous verification key, run key generation again and then issue a new verification key for the user. However, depending on the PKI in use, this may be expensive.

It is possible to design a password change protocol that only changes the server's signing key share, not the verification key. Such a protocol may not be a good idea, however. If we consider a corrupt server that has somehow discovered the user's password, it will be impossible for the user to recover by changing his password, even if the new password is hard to guess.

We also note that anyone who gets access to the messages sent in either the key generation phase or the signing protocol will learn enough to be able to search for the user's password. Specifically, this means that both protocols should only run over secure networks, and that the server's issue functionality is protected.

References

[1] Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT*, volume 1163 of *Lecture Notes in Computer Science*, pages 244–251. Springer, 1996.

[2] Giuseppe Ateniese and Breno de Medeiros. A provably secure Nyberg-Rueppel signature variant with applications. Cryptology ePrint Archive, Report 2004/093, 2004. `http://eprint.iacr.org/`.

[3] Jan Camenisch, Jean-Marc Piveteau, and Markus Stadler. Blind signatures based on the discrete logarithm problem. In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 428–432. Springer, 1994.

[4] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO*, pages 199–203. Plenum Press, New York, 1982.

[5] Ivan Damgård and Gert Læssøe Mikkelsen. On the theory and practice of personal digital signatures. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 277–296. Springer, 2009.

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

[7] Kristian Gjøsteen and Øystein Thuen. Password-based signatures. In Svetla Petkova-Nikova, Andreas Pashalidis, and Günther Pernul, editors, *EuroPKI*, volume 7163 of *Lecture Notes in Computer Science*, pages 17–33. Springer, 2011.

[8] Laurie Haustenne, Quentin De Neyer, and Olivier Pereira. Elliptic curve cryptography in JavaScript. Cryptology ePrint Archive, Report 2011/654, 2011. `http://eprint.iacr.org/`.

[9] Yong-Zhong He, Chuan-Kun Wu, and Deng-Guo Feng. Server-aided digital signature protocol based on password. In *Security Technology, 2005. CCST 2005*, pages 89–92, 2005.

[10] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 1997.

[11] Peter Landrock. New PKI protocols using tamper resistant hardware. In Stig Fr. Mjølsnes, Sjouke Mauw, and Sokratis K. Katsikas, editors, *EuroPKI*, volume 5057 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2008.

[12] Kaisa Nyberg and Rainer A. Rueppel. A new signature scheme based on the DSA giving message recovery. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 58–61. ACM, 1993.