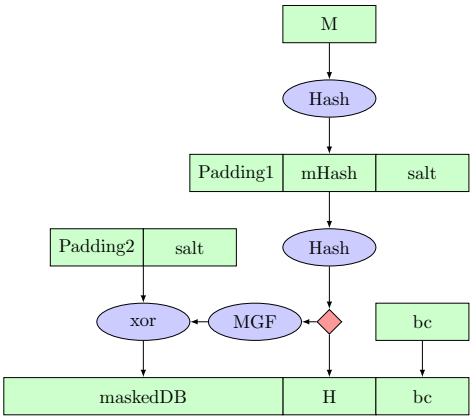
RSA-PSS – Provable secure RSA Signatures and their Implementation

Johannes Böck
http://rsapss.hboeck.de/

May 4, 2011



Diplomarbeit

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

Eingereicht von Johannes Böck
geboren am 30.06.1980 in Backnang

Betreuer / 1. Gutachter: Prof. Dr. Ernst-Günter Giessmann
http://www.informatik.hu-berlin.de/~giessman/
2. Gutachter: Prof. Dr. Johannes Köbler
http://www.informatik.hu-berlin.de/~koebler/

Contents

1 Introduction

Public key cryptography and digital signatures are an important building block of today's computer security. However, the theoretical foundations of their security are quite weak. The whole security relies on unproven assumptions. For high or long term security requirements this is particularly unsatisfying.

It has been shown in the past that unexpected breakthroughs in cryptanalysis happen. Algorithms that were considered secure can be broken (MD5, SHA-1, RSA with 512/768 bit). As cryptanalysis advances, it is advisable to implement extra security measures against security threats yet unknown.

The RSA algorithm is the most popular asymmetric public key algorithm. It can be used for both signing and encryption. For both security and performance reasons, RSA can not be used in its "plain" form, it needs some kind of preprocessing for the messages. For signatures, this is traditionally done with a hash-function and some fixed padding.

While this has no known flaws, it is likely that this is not as secure as it could be. In 1996, Bellare and Rogaway suggested two "provable secure" schemes for RSA: The Probabilistic Signature Scheme (PSS) for signatures and Optimal Asymmetric Encryption Padding (OAEP) for encryption. These provide "provable" security under certain model assumptions. This means the security of the scheme can be directly related to the security of the RSA function and the used hash function. They also add randomization, which strengthens the algorithms against certain attack scenarios.

In this work, I will investigate the use of the Probabilistic Signature Scheme (PSS) for RSA. I will give an overview of reasons why it is a security improvement over earlier schemes. I will then investigate the use of PSS in real-world applications and analyze why it has not been adopted widely yet.

Chapter 2 provides some theoretical background about RSA, public key encryption and the idea of provable security within a theoretic model (the random oracle model). Chapter 3 will then give an overview of the development and standardization process of RSA-PSS. In chapter 4, we will see that RSA-PSS not only protects against unknown attack scenarios, but that the design also lowers the possibility of certain implementation flaws that happened in the past within RSA implementations. We will then have a look at the differences between the original PSS proposal from 1996 and the later standardized version in chapter 5. We will see that a questionable trade-off on security has been made to make implementation easier.

Afterwards, we will focus on the practical use of RSA-PSS. Chapter 7 will give an overview of the use of RSA-PSS within the most popular protocols involving cryptography used on the Internet today. As part of that, I also created an implementation of RSA-PSS signatures for X.509 in the widely used nss library (chapter 8) in the Google Summer of Code 2010. I have setup some online test cases to check the RSASSA-PSS capabilities of X.509 implementations within web browsers (chapter 9). Afterwards, in chapter 10 we will have a

look at requirements by law and possible transition timelines by standardization bodies and other involved parties.

In chapter 11, we will give an outlook on possible future developments in the field of provable security and ask the question if it is possible to have really provable security that does not depend on model assumptions. Finally, I will draw some conclusions from the work done.

2 Theoretical Background

2.1 Public Key Cryptography and RSA

Public key cryptography, also often called asymmetric cryptography, is the idea of signature and encryption systems that work with a public and a private key. The public key can be used to encrypt messages and only a person with access to the private key is able to decrypt it. For signatures, the private key is used to generate a signature for a document which can then be verified with the public key. Encryption and signatures are sometimes done with distinct algorithms, sometimes the same algorithm can be used for both.

It is assumed that an attacker has access to the public key. The attacker also knows how the algorithm itself works (this is called Kerckhoffs' Principle). The only thing that is secret is the private key.

Up to 1977, it was questionable if public key cryptography was possible at all (and this has not finally been decided yet, see chapter 11). In a ground-breaking article titled "New Directions in Cryptography", Whitfield Diffie and Martin Hellman presented the general idea of using so-called trap-door functions for public key cryptography [Diffie and Hellman, 1977] . A trap-door function is a special kind of one-way function. One-way functions are easy to calculate in one direction, but difficult in the other. To be suitable for public key cryptography, trap-door functions need another ability: It should be hard for an attacker to invert a calculation, but it should be easy for the owner of a special secret (the private key) to invert the calculation. Shortly after the paper from Diffie and Hellman, Ron Rivest, Adi Shamir and Leonard Adleman presented the RSA algorithm [Rivest et al., 1977] , the first public key algorithm. RSA remains secure up until today and is still by far the most frequently used public key algorithm.

2.2 How RSA works

This chapter will not cover all the details of RSA, we will just try to get a basic understanding how RSA encryption and signatures look like.

An RSA key consists of three elements: A modulus N, a public exponent e and a private exponent d. The modulus N is a large number that is a product of two primes p and q ($N = p \cdot q$). All calculations are done with modulus

reduction, meaning every result that is bigger than N is divided by N and the remainder is taken.

The exponents are chosen in a way that for any number M with $M < N$, the following is always true:

$$M = M^{d \cdot e} \mod N = M^{e \cdot d} \mod N \quad (1)$$

The public key is the pair of (N, e), the private key is the pair of (N, d). We generally can assume that exponentiation in a modulus can be done fast.

Within the key generation, it is possible (and necessary) to generate the private exponent d with the knowledge of e and the factors p and q. d and e have the relation

$$d \cdot e \equiv 1 \pmod{(p-1)(q-1)}$$

This is true due to the Chinese remainder theorem and Fermat's little theorem. This also means that if an attacker is able to generate the factors p and q out of N, the attacker can generate the private key. So the security of RSA relies on the fact that factoring N is a hard problem.

If Bob wants to send a message to Alice, he needs her public key (N, e). The encryption of a message M (with $M < N$) is done by the operation $E = M^e \mod N$. If Alice wants to decrypt the message, she calculates $M_D = E^d \mod N$. As we can easily see with equation (1), M_D is exactly M, because $M_D = (M^e)^d \mod N = M^{(e \cdot d)} \mod N = M$.

Now we will have a look at signatures. Alice wants to sign a message M so that Bob can verify with her public key that the message truly is from her. She calculates $S = M^d \mod N$. She then sends M and S to Bob. Bob can calculate $M_V = S^e$. Again, with equation (1) it can be easily seen that $M_V = M^{d \cdot e} = M$. If this is the case, Bob can be sure that S was created with Alice's private key.

2.3 RSA and Factoring

As we have already seen, public key algorithms are based on trap-door functions, a special kind of mathematical one-way functions. The RSA algorithm is based on factoring. It is easy to multiply two large prime numbers, but no algorithm is known that is able to factorize a large number efficiently.

However, it is not proved that RSA is as secure as factoring. It can be shown that if an attacker is able to generate a private key from a public key, he is also able to factorize large numbers. But until today nobody was able to prove that an attacker who is able to decrypt messages or forge signatures is also able to factorize large numbers. So it is unknown if the complexity of the RSA problem is the same as the complexity of factoring.

In 1979, two years after the publication of RSA, Michael O. Rabin proposed the Rabin public key algorithm [Rabin, 1979] . The Rabin algorithm has the advantage of being provably as secure as factoring. It has, however, a down-

side: Every decryption operation produces four possible outputs and thus is not suitable for practical applications. Williams suggested a change that avoids this ambiguities [Williams, 1980] . This is called the Rabin-Williams algorithm, abbreviated RW. We will come back to discuss Rabin-Williams in chapter 3.6.

The original Rabin algorithm already included the idea of hashing a message before signing - a concept which became also crucial to make RSA secure, as we will see in the next chapter.

2.4 Plain / Textbook RSA

In the algorithm defined in the original RSA paper [Rivest et al., 1977] , the function for signing is $S = M^e \mod N$, for verification $M = S^d \mod N$ (S: Signature, M: Message, e, N: public key, d, N: private key). This original version – often called textbook RSA – has security problems. Assuming an attacker has messages M_1 and M_2 and signatures $S(M_1)$ and $S(M_2)$. Due to the nature of the RSA signature function, $(S(M_1) \cdot S(M_2)) \mod N$ would give a valid signature for $(M_1 \cdot M_2) \mod N$. This is called the multiplicative property of the RSA algorithm [Davida, 1982] .

Another problem of the original, unpadded RSA scheme is the fact that the encryption operation is directly the inverse of the signature operation. Thus, if a person is using the same RSA key pair for both signing and encryption, an attacker might be able to give the person encrypted data and asks for a signature. If the victim signs, the signature contains the decrypted data. While this may sound like an unrealistic threat, signature generation is often part of an automated system.

2.5 Hash Functions

A way to avoid attack vectors against plain RSA is the use of a cryptographic hash function. A hash function is a one-way function mapping any input to an output of a fixed length. A cryptographic hash function can be classified by certain properties. The most important one is collision resistance: A hash function is called collision resistant if it is not possible for an attacker to generate two different inputs which produce the same output in a reasonable amount of time (in mathematical terms, it shall not be possible to efficiently generate M_1 and M_2 so that $M_1 \neq M_2$ and $\mathit{Hash}(M_1) = \mathit{Hash}(M_2)$).

If for a given hash H and a message M_1 with $H = \mathit{Hash}(M_1)$ an attacker is able to create a message M_2 with $\mathit{Hash}(M_1) = \mathit{Hash}(M_2)$, this is called a second preimage attack. If for a given hash H an attacker is able to generate a message M so that $H = \mathit{Hash}(M)$, this is called a preimage attack.

For simple signature schemes, it is necessary to have a hash function which is collision resistant. We will later see that in certain scenarios a preimage resistant hash function may be enough.

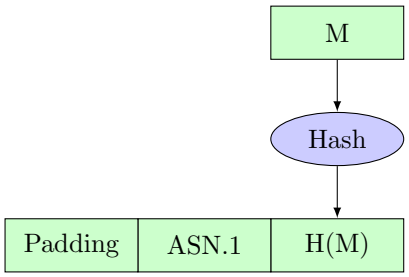


Figure 1: PKCS #1 v1.5 hash-then-sign padding [RSA Inc., 1993]

For a long time, the most common hash functions were MD5 – developed by Ron Rivest, one of the inventors of RSA – and SHA-1, developed by the US NIST (National Institute of Standards and Technology) and the NSA.

In recent years, the research on the security of hash functions has brought a number of unexpected results. The security of the two most commonly used hash functions MD5 and SHA-1 has been seriously damaged. In 2004, the Chinese researcher Xiaoyun Wang and her team were able to generate collisions for MD4, MD5 and RIPEMD [Wang et al., 2004] . For SHA-1, the same team showed an attack with a complexity of 2^{69} [Wang et al., 2005] . This is not practical on common hardware, but it can be expected that an attacker with a reasonable amount of money to build special purpose hardware is able to perform this attack. Later, Wang's team was able to improve the attack to a complexity of 2^{63}.

The SHA-2 hash function family (SHA-224, SHA-256, SHA-384 and SHA-512), also developed by the US NIST and the NSA, remains secure until today.

After the findings of Wang, the NIST announced a competition for a new hash standard SHA-3. 64 submissions were made to the competition[1], five are currently left for the final round (Skein[2], Keccak[3], JH[4], Grøstl[5], and BLAKE[6]). The winner will be announced in late 2012.

2.6 Hash-then-Sign

Hash-then-sign has been the most common way to prevent most of the problems with textbook RSA for signatures. The cryptographic hash of the input message is calculated, together with some fixed padding this is used as an input for the RSA function.

With slight modifications, the hash-then-sign method is still used in the vast

[1] http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo
[2] http://skein-hash.info/
[3] http://keccak.noekeon.org/
[4] http://www3.ntu.edu.sg/home/wuhj/research/jh/
[5] http://www.groestl.info/
[6] http://131002.net/blake/

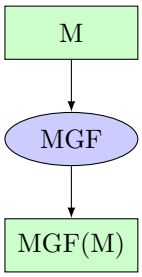


Figure 2: Full Domain Hashing

majority of today's applications. The standard PKCS #1 v1.5 [RSA Inc., 1993] uses a combination of a prefix (00 01 ff ff ... ff 00), an identifier for the hash function and the hash itself as an input for the RSA function. There must be at least eight ff bytes in the padding.

Common cryptographic hash functions today have a size between 256 and 512 bits. Due to the efficiency of the best factoring algorithms today, reasonable key sizes for RSA are between 2048 and 4096 bits. Let's assume a 2048 bit RSA key with a SHA-256 hash. While the RSA function accepts an input of 2048 bit length, only 256 bits are really used, the rest is fixed. It seems reasonable to choose a scheme that uses the full possible input length of 2048 bits. A possibility would be the creation of a hash function with the same size as the key. It is easy to create a hash function which takes an additional argument for the output length out of any other hash function. Such a function is called mask generation function (MGF). A simple approach for such a function would be the use of multiple hashes of the input concatenated with a counter variable. Using such a function to prepare the input message has been proposed under the name Full Domain Hashing (FDH) [Bellare and Rogaway, 1996] , which is already an improvement over the old hash-then-sign schemes.

2.7 Provable Security

With advances in cryptographic research, it was desirable to provide "provable" security. Pretty much all cryptographic methods today rely on assumptions. It is in general very difficult to prove any lower complexity bounds for mathematical problems. Therefore it is also difficult to define lower complexity bounds for breaking any cryptographic algorithm. The trust in cryptographic algorithms is almost entirely based on the assumption that if publicly documented and well-known algorithms have been reviewed by a large number of researchers without any breakthroughs in attacks, the algorithm is believed to be secure.

So provable security can make no statements about the security of a complete cryptographic system. It can however make statements in the way that the security of a system can be related to the security of an underlying fundamental problem.

2.8 Random Oracle Model

The random oracle model is a theoretic model assuming an ideal hash function [Bellare and Rogaway, 1993] . The hash function works like a black box that returns a truly random output for every new input. The oracle "remembers" all inputs and if the same input is given, it produces the same output.

There are some theoretic doubts about security proofs in the random oracle model and some researchers remain sceptical how relevant a proof in the random oracle model is for security in reality [Dent, 2006] . Canetti was able to construct a special scheme that is provable secure in the random oracle model while at the same time being insecure with any real hash function applied [Canetti et al., 2002] . But even to those sceptical researchers a proof of a scheme in the random oracle model is considered a vast improvement compared to no proof at all.

2.9 Randomization / Salt

Randomization means that some part of an algorithm has a random input, which causes the output to be different for equivalent inputs. In the most commonly used RSA schemes from the standard PKCS #1 v1.5, the encryption scheme contains randomization. This is crucial to prevent dictionary attacks. In scenarios where only a limited number of messages is possible (for example a yes/no reply inside a protocol), an attacker could simply try out all possibilities and compare them to the encrypted message.

However, for signatures, the PKCS #1 v1.5 hash-then-sign scheme is strictly deterministic. For the same input and key, the output will always be the same. Including randomization is generally considered as a strengthening of a cryptographic protocol, which we will demonstrate later in chapter 4.2.

Randomization in a protocol is usually done with a so-called salt. A fixed-size random value that is included at some point in the protocol – often before a hashing-step. The salt usually then has to be shipped along or in some encoded form within the protocol. Sometimes the salt is also called seed or nonce.

2.10 Other Public Key Algorithms

Beside RSA, only a very small number of public key algorithms exist. Also quite common is the ElGamal algorithm [ElGamal, 1985] , which relies on the difficulty of the discrete logarithm problem. But similar to RSA, there is no proof that breaking ElGamal is as hard as solving the discrete logarithm problem. ElGamal is often found under the name DSA (Digital Signature Algorithm), a standard by the US National Institute of Standards and Technology (NIST).

It is generally believed that solving a discrete logarithm is of similar difficulty as factoring large numbers. The most efficient algorithms for solving both are the same. So one can generally assume that ElGamal and RSA with the same

key size have equal security. However, it has never been proven that factoring and discrete logarithms have the same complexity.

Another variant of public key cryptography that has gained popularity in recent years is elliptic curve cryptography. Elliptic curve cryptography also relies on the ElGamal / DSA algorithm, thus the most widely used algorithm is called ECDSA. ElGamal can be defined over any cyclic group. Usually, it is done over a prime field. But the points on a so-called elliptic curve also build a cyclic group. It is generally believed that using ElGamal over elliptic curves can yield the same security with much smaller keys. The reason for that is that the best known algorithms for solving the discrete logarithm problem in prime fields and for integer factorization like the general number field sieve do not work for solving discrete logarithms within elliptic curves. The most efficient algorithm for solving discrete logarithms in elliptic curves is Pollard's rho algorithm. Common key sizes for ECDSA are between 160 and 571 bit, compared to a safe key size between 2048 and 4096 for RSA / ElGamal based algorithms.

It should be noted however that the security of elliptic curves also relies on unproven assumptions. There is no proof that it is not possible to extend the general number field sieve to discrete logarithms in elliptic curves. It also may be that a yet unknown algorithm is able to break elliptic curve cryptography for equal key sizes [Schneier, 1999] .

While a number of other public key algorithms have been proposed in the past, most of them only gained academic interest. We already mentioned the Rabin [Rabin, 1979] and Rabin-Williams [Williams, 1980] cryptosystems, which are provably as secure as factoring. A lot of public key systems proposed in the past have been broken (for example the Merkle-Hellman knapsack cryptosystem [Shamir, 1984] or the SFLASH signature scheme). Others remain impractical, like the McEliece system [McEliece, 1978] , which has public and private keys of several hundred kilobytes. Another cryptosystem with interesting properties is NTRU, but it is rarely used because it is patented. McEliece and NTRU are possible candidates for public key cryptosystems resistant to quantum computers [Bernstein, 2009] .

2.11 Summary

We have introduced the concept of public key cryptography in general. We have discussed the RSA algorithm with the hash-then-sign scheme, which is what the vast majority of RSA implementations are using today.

We have discussed the idea of provable security, which can – with today's knowledge – only work under certain model assumptions.

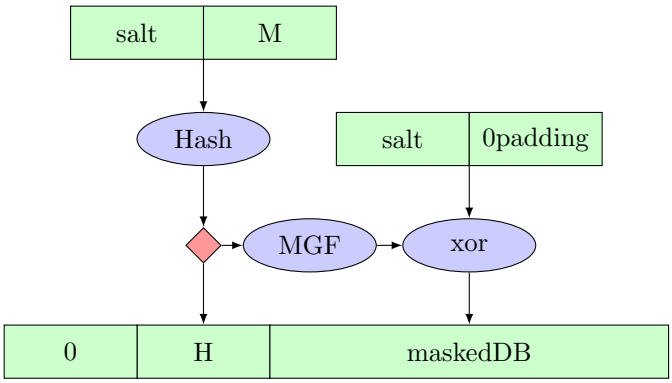


Figure 3: Original PSS algorithm [Bellare and Rogaway, 1996] [7]

3 The Probabilistic Signature Scheme

After inventing the random oracle model and proposing a scheme they later called Full Domain Hashing, Bellare and Rogaway continued their work by proposing two new schemes for RSA that became the basis of upcoming cryptography standards. For encryption, they propose the Optimal Asymmetric Encryption Padding (OAEP) [Bellare and Rogaway, 1995] .

In a paper from 1996, Rogaway and Bellare discuss the security of the original hash-then-sign scheme and the Full Domain Hashing scheme. They come to the conclusion that for both schemes, no tight security assertions can be made. They provide another scheme, the Probabilistic Signature Scheme. They can prove in the random oracle model that the security of the scheme can be tightly related to the hardness of inverting the RSA function [Bellare and Rogaway, 1996] .

Both PSS and OAEP have randomization, which provides protection against certain types of implementation attacks (see chapter 4.2). It should also be noted that the randomization is a crucial part of the security proof and that attacks may be possible if the source of the random numbers is weak [Brown, 2005] .

3.1 How PSS works

PSS takes the input message and a salt (a random number) and runs them through a hash function. This hash H is used as the beginning part of the output. Then, a mask of H is calculated, which has the length of the RSA modulus minus the length of H. This mask is then XOR-ed with the salt (and some zero padding) and the output will be called $maskedDB$. Then, $maskedDB$ is appended to H to generate the input for the RSA function (see figure 3 for a graphical representation of the algorithm).

[7] The diagram in the original PSS paper looks quite differently, I created this diagram in a similar way as the one in PKCS #1 v2.1 to outline the similarities and differences between the original proposal and the final standard.

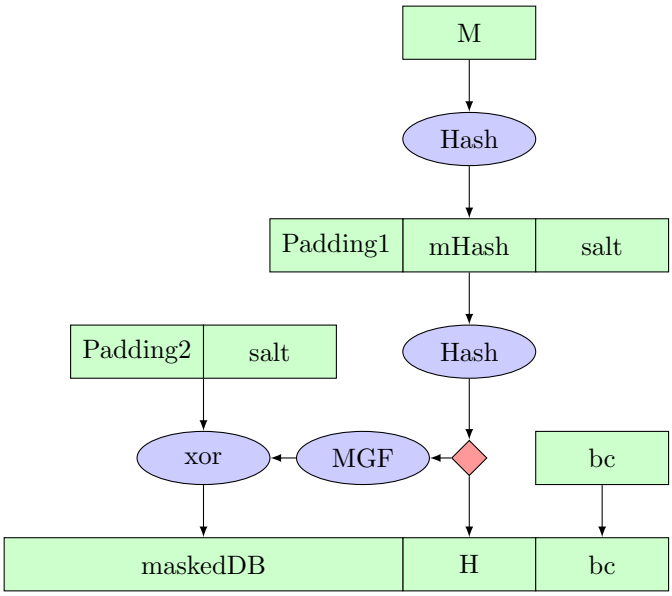


Figure 4: RSASSA-PSS according to PKCS #1 v2.1 / RFC 3447 errata[8]

The variant that later became part of standards is slightly different. H and $maskedDB$ are switched in their order. The input message M is hashed at the beginning and then hashed again with a salt appended (see figure 4 for a graphical representation). We will discuss the security impact of those changes later in chapter 5.

It is in theory possible to set the salt size to zero. This makes PSS a deterministic algorithm again which has security properties similar to Full Domain Hashing.

3.2 Appendix and Message Recovery

PSS comes in two variants, with appendix and with message recovery. Signature scheme with appendix (sometimes referred to as SSA) means that the signature is an additional block of data appended to a signed message. Message recovery means that parts of the message are encoded within the signature. This can be relevant if the size of the transmitted data is a bottleneck and every byte matters – for example on smart cards. In the vast majority of applications, message recovery is not needed and signature scheme with appendix is the default and also the only variant of PSS specified in the PKCS #1 standard by RSA Inc. and the IETF (Internet Engineering Task Force).

[8]The diagram in the original version of PKCS #1 v2.1 / RFC 3447 is erroneous, however, the IETF never changes finished RFCs. The correct version of the diagram is listed in the RFC Errata at `http://www.rfc-editor.org/errata_search.php?rfc=3447`

```
Signature Algorithm: rsassaPss
     Hash Algorithm: sha512
     Mask Algorithm: mgf1 with sha512
     Salt Length: 01BE
     Trailer Field: 0xbc (default)
```

Listing 1: "PSS parameter block in OpenSSL"

3.3 Standardization of Algorithm Primitives

In 1998, Bellare and Rogaway submitted the PSS algorithm to the IEEE (Institute of Electrical and Electronics Engineers) P1363 working group for public key cryptography. In 2000, the IEEE released P1363a [IEEE, 2004] , which contains a PSS based (but significantly modified, see chapter 5) RSA signature algorithm padding called EMSA4 and also OAEP based RSA encryption.

The PKCS standards are provided by RSA Inc., the company of RSA founder Ron Rivest. PKCS #1 contains primitives for public key operations. In 2002, RSA laboratories published PKCS #1 v2.1 [RSA Inc., 2002] , which provides cryptographic primitives for RSA-PSS and RSA-OAEP. The function EMSA-PSS in PKCS #1 v2.1 is compatible with EMSA4 in IEEE 1363A. The PKCS standards are widely used within the development of cryptographic solutions and standards. In 2003, the IETF republished PKCS #1 v2.1 as RFC 3447. RFC stands for "Request for Comment", all standards by the Internet Engineering Task Force (IETF) that build the basis of the Internet are called RFCs.

The PKCS #1 v2.1 standard also specifies so-called Object Identifiers (OIDs) for PSS and OAEP. Object Identifiers are part of the ASN.1 encoding standard and provide a hierarchical standard for unique identifiers for objects. All ASN.1-based cryptographic standards contain unique algorithm identifiers as OIDs.

id-RSAES-OAEP	1.2.840.113549.1.1.7
id-mgf1	1.2.840.113549.1.1.8
id-pSpecified	1.2.840.113549.1.1.9
id-RSASSA-PSS	1.2.840.113549.1.1.10

id-mgf1 is for the mask generation function MGF1. pSpecified is only used by OAEP, we will not cover the details here.

With old PKCS #1 v1.5 paddings, there was usually an OID for every combination of algorithm and hash function, so for example 1.2.840.113549.1.1.11 is the OID for sha256WithRSAEncryption. This is different with PSS/OAEP. Instead, we have a parameter block after the algorithm identifier containing the hash function, the mask generation function, the hash function used to generate the mask generation function, the salt length and a trailing field (the trailing field has no real purpose, as it has a fixed value, it is only there for compatibility reasons with IEEE P1363a). Listing 1 shows an example of the parameter block in OpenSSL.

In the PKCS standard, the old and new signature schemes are called RSASSA-PSS and RSASSA-PKCS1-v1_5, the underlying encoding operations EMSA-PSS and EMSA-PKCS1-v1_5. These naming conventions will be used within this document. If we refer specifically to the original PSS version, we will call it PSS96.

3.4 Mask Generation Function

PSS requires a so-called mask generation function. This is basically like a hash function, but with a variable output size. In other contexts, similar functions are also called key derivation functions. The PKCS #1 v2.1 standard lists only one possible function, MGF1. It is based on an existing hash algorithm and just works by using the input plus a four byte counter starting with zero as an input for the hash function and increment the counter to get enough output bits from the hash function. The last output is cut to get the required size. MGF1 is mostly equivalent to the key derivation function KDF2, as specified in ISO 18033-2 [ISO, 2006] .

MGF1 has the property that two calls to MGF1 with the same hash function and the same input with a different output size would lead to an output identical at the beginning. For example, if we calculate both `MGF1(SHA-256, "hello", 10)` and `MGF1(SHA-256, "hello", 20)`, we get:

```
MGF1(SHA-256, "hello", 10) = da75447e22f9f99e1be0
MGF1(SHA-256, "hello", 15) = da75447e22f9f99e1be09a00cf1a07
```

As we see, the first 10 bytes of the second MGF1 output are identical to the first MGF1 output.

This looks like an unideal cryptographic property of a mask generation function. This has no impact on PSS – the only possible scenario where one would have the same input to MGF1 and a different output length would be if one used the same salt and message with a different key size (compare figure 4). So within a proper implementation of RSASSA-PSS, the use of MGF1 is just fine. However, if MGF1 is used within other cryptographic constructions, this property should be consider and it should be investigated if it causes any problems.

3.5 Patents on PSS

The University of California has filed two patents on PSS [Bellare and Rogaway, 2001] [Bellare and Rogaway, 2006] . During the standardization process for IEEE P1363a, they claimed that if PSS with appendix gets standardized, they will freely license the patent to anyone doing implementations of the standard [Grell and University of California, 1999] . They would however charge fees for PSS with message recovery.

Both patents have not been renewed and thus #6,266,771 expired on July 24, 2009 and #7,036,014 expired on April 25, 2010. So currently, PSS in any

variant is most probably free of patent claims (source: personal email from the University of California, Office of Technology Transfer).

I am not aware of any patents covering OAEP.

Due to the free licensing for PSS with appendix since 2000 (the finalization of IEEE P1363a), it is unlikely that the patent situation was hindering the implementation and use of PSS based solutions.

3.6 PSS for Rabin-Williams

We mentioned in chapter 2.3 that RSA is based on the factoring problem, but nobody has been able to prove that breaking RSA is as hard as factoring. Talking about provable security, it would be nice to have such a statement.

When developing PSS, Bellare and Rogaway also considered the Rabin-Williams algorithm, which is provably as hard as factoring. They provide a variant of PSS for it. In terms of provable security, RW-PSS would definitely be the better choice, as it not only proves that there are no flaws within the padding scheme, but also that the security of the cryptosystem itself can be directly related to a well-known basic mathematical problem.

Despite its good security properties, RW is rarely used today. Primitives are standardized within IEEE P1363a, but not within PKCS #1 and not in any RFC. No major high-level protocol involves support for RW signatures. So the only reason not to use the RW algorithm today are missing standards. From a cryptographic point of view it would make a lot of sense.

3.7 Summary

Based on the idea of provable security in the random oracle model, Philipp Rogaway and Mihir Bellare developed the Probabilistic Security Scheme (PSS). With PKCS #1 v2.1, we have a widely accepted standard of cryptographic primitives for RSA with PSS. Although patented in the past, there are no patent claims today to prevent the implementation of PSS.

The more secure combination of the Rabin-Williams algorithm and PSS (RW-PSS) is barely used and is missing widely accepted standards.

4 Attacks on old Signature Schemes

As stated before, there are no known security flaws in the old RSASSA-PKCS1-v1_5 signature scheme. In this chapter, we will discuss two attacks on weak implementations of RSASSA-PKCS1-v1_5 that would not have been possible with RSASSA-PSS. The purpose is to show that PSS is generally more robust against implementation flaws.

```
--- crypto/rsa/rsa_sign.c 26 Apr 2005 22:07:17 -0000  1.21
+++ crypto/rsa/rsa_sign.c 4 Sep 2006 15:16:57 -0000
@@ -185,6 +185,23 @@
     sig=d2i_X509_SIG(NULL,&p,(long)i);

     if (sig == NULL) goto err;
+
+    /* Excess data can be used to create forgeries */
+    if(p != s+i)
+      {
+      RSAerr(RSA_F_RSA_VERIFY,RSA_R_BAD_SIGNATURE);
+      goto err;
+      }
+
+    /* Parameters to the signature algorithm can also be used to
+       create forgeries */
+    if(sig->algor->parameter
+       && sig->algor->parameter->type != V_ASN1_NULL)
+      {
+      RSAerr(RSA_F_RSA_VERIFY,RSA_R_BAD_SIGNATURE);
+      goto err;
+      }
+
     sigtype=OBJ_obj2nid(sig->algor->algorithm);
```

Listing 2: Patch for OpenSSL to prevent Bleichenbacher attack

4.1 Bleichenbacher Attack on PKCS #1 v1.5 Implementations

At the rump session of Crypto 2006, Daniel Bleichenbacher presented an attack against certain implementations of RSA with the PKCS #1 1.5 hash-then-sign scheme (the attack has been written down by Hal Finney [Finney, 2006]). The attack makes use of the rather simple structure of the RSA function's input and the unused padding part.

The message encoding in EMSA-PKCS1-v1_5 looks like this:

00 01 FF FF ... FF 00 $\|$ ASN.1 $\|$ $H(M)$

ASN.1 contains just an ID of the used hash algorithm, $H(M)$ is the hash of the input message M. Several implementations did not calculate the length of the padding, instead they scanned the FFs until they found a zero byte. This caused an input looking like this to be considered valid, too:

00 01 FF FF ... FF 00 $\|$ ASN.1 $\|$ $H(M)$ $\|$ garbage

With a small RSA exponent (the attack was shown for $e = 3$), by carefully selecting "garbage", it is possible to construct the input to be a perfect cube, which makes it possible to use the cube root as a valid signature.

In listing 2 you can see the patch to prevent the Bleichenbacher attack in OpenSSL[9] (CVE-2006-4339, fix included since OpenSSL version 0.9.7k and 0.9.8c). `s` is a pointer to the beginning of the signature, `p` is the current pointer, which should be at the end of the signature and `i` is the expected size of the signature. Therefore, the check for `(p != s+i)` guarantees that no garbage is present after the message hash.

It should be stressed that this attack is not on RSA or PKCS #1 v1.5 itself, but on faulty implementations. A check like the one shown in OpenSSL to verify that either the padding has the correct length or that there is no garbage data after the hash is a sufficient counter-measure against this threat. When it was first discovered, a number of popular RSA implementations were vulnerable to the attack.

The attack vector here is that large parts of the RSA input in EMSA-PKCS1-v1_5 are fixed values. With EMSA-PSS, the whole encoded message that is passed to the RSA function depends on the input message, so this kind of attack would not have been possible.

It should also be noted that the whole attack depends on a very small exponent e of the RSA key, which is considered bad practice and may pose other threats.

4.2 Fault-based Attack

In 1996, Dan Boneh and others [Boneh et al., 1996] presented an attack on RSA doing faulty calculations. By injecting random faults into the calculations of RSA, they are able to regenerate the private key from the knowledge of the faulty signatures. RSA implementations using the Chinese remainder theorem to speed up calculations are especially vulnerable – a single erroneous signature allows the regeneration of the private key.

Similar attacks have been shown subsequently. For example, Eric Brier and others [Brier et al., 2006] provide a fault-based attack on the RSA modulus operation. The assumption is that an attacker is able to flip random bits in the modulus. After a certain number of faulty signatures (approx. 60.000), an attacker is able to calculate the private key with statistical analysis of the signatures.

Protection against fault-based attacks like this is especially important in embedded devices like chip cards that are built not to expose the private key, but to provide cryptographic operations like signatures in an environment potentially under control of an attacker. The easiest counter-measure without changing the algorithms involved is to always verify created signatures and not to expose any faulty results.

However, the attack relies on the fact that the attacker knows the full input of the RSA function. Coron and others [Coron et al., 2009] investigated the

[9] `http://www.openssl.org/news/patch-CVE-2006-4339.txt`

impact of fault-based attacks on randomized cryptographic schemes. They were able to extend the Boneh attack to certain randomized schemes, but only in schemes that expose the random input to the attacker. In PSS, the random salt is encoded within the signature – it is not possible to gain the salt from a faulty signature. In a later work, Coron and Mandal [Coron and Mandal, 2009] were able to prove that PSS is invulnerable against these kinds of fault-based attacks.

As we have seen, similar to the Bleichenbacher attack, fault-based attacks are no direct attacks against old padding schemes. A careful implementation that guarantees that an attacker will never see a faulty signature can avoid them. But PSS would have prevented the whole attack in the first place, without any extra security measures. This shows again that PSS is much more robust with respect to implementation problems.

4.3 Summary

We have seen two examples of practical attacks that make use of specific aspects of the hash-then-sign schemes. Both attacks would have been avoided with the design of PSS. So we have seen that PSS provides more robustness against real-world problems.

Both examples were attacks not against the algorithms itself, but against their implementations. Therefore, PSS is not only a protection against yet unknown security flaws, but it also seems that it makes implementation flaws less likely.

5 Input hashing

There are some substantial differences between the original proposal [Bellare and Rogaway, 1998] and the final standard [RSA Inc., 2002] , [IEEE, 2004] .

We will first have a look at an attack that was done against X.509 / SSL certificates and then discuss the impact on both PSS variants.

5.1 Real-world Attack on X.509 Certificates using MD5

The MD5 hash function was the de facto standard for a cryptographic hash function for quite a while. In 2004, it got under serious attack – a collision was shown by Xiaoyun Wang and others [Wang et al., 2004] based on previous work by Hans Dobbertin [Dobbertin, 1996] .

A collision of a cryptographic hash function means that you can generate two inputs A and B with $Hash(A) = Hash(B)$. There was, however, some debate about the impact of a collision, as many uses of cryptographic hash functions are only affected by so-called preimage attacks. A preimage attack means that

for a given output O of a hash function you can generate a value A so that $Hash(A) = O$.

Improvements on the attack on MD5 allowed not only random inputs with a collision, but also different meaningful inputs generating the same output, assuming you have some way to put random looking, specifically crafted bits into the input. At the 25th Chaos Communication Congress, a group of researchers presented a way how they created a real-world working CA certificate from RapidSSL. RapidSSL is a certificate authority accepted by all major web browsers. It would've been accepted by all major browsers, but they (voluntarily) created it with an already expired date [Sotirov et al., 1998] . This was not a limitation of the attack, but a measurement by the security researchers to avoid misuse of their results.

The main problem they faced was that they had to predict the exact input of the certificate that the CA would put into their signing mechanism. It was only possible because RapidSSL generated all certificates with a timestamp exactly 6 seconds after certificate request and used incremental serial numbers. By doing several tests they were able to predict the number of issued certificates with a high probability and thus where able to prepare a collision for the correct serial number.

5.2 Differences between original and standardized PSS

The first step of the original proposal was to generate a salt[10] and hash both the salt and the message M:

$Hash(padding||salt||M)$

Contrary to that, PKCS #1 v2.1 (and also IEEE P1363a) uses a hash of the message as the first step and then adds the salt:

$mHash = Hash(M)$
$M' = Hash(padding||M||salt)$

Please note also that the original proposal puts the salt in front of the message, while the standard appends the salt to the end.

While this change does not look very significant, it seriously weakens the security when using a real-world hash function. This can be easily seen when comparing that to the attack presented above. We assume we have a broken hash-function that allows the generation of collisions (examples would be MD2, MD5, RIPEMD, SHA-0), but is still resistant to preimage attacks. The PKCS #1 v2.1 variant would allow us to do the following: We create two inputs, one looking nice and one malicious with the same hash sum. We ask the owner of the private RSA key to sign the nice input. The signature would still be valid for the malicious input (exactly what has been done in the RapidSSL attack).

[10]The PSS96 proposal uses the term seed instead of salt. For clarity, we will always use salt within this document.

This only works because the first thing done in the whole process is hashing the message and this is the only place in the whole algorithm where the message is used at all. With the original algorithm, this would be avoided, as the attacker cannot precompute a collision of $salt||M$ when he does not know the salt.

For this to have an effect, it is crucial that the salt is put in front of the message. This is due to the nature of most real-world hash functions (including MD5, SHA-1, SHA-256) - they are all based on the Merkle-Damgard design. They have an internal state. If we have a collision $Hash(M_1) = Hash(M_2)$, then this also implies $Hash(M_1||salt) = Hash(M_2||salt)$. At the point when the hash function processes the salt, the internal state of the hash function is identical and with an identical input following, both will produce the same output.

PKCS #1 v2.1 (chapter 9.1, page 37 in [RSA Inc., 2002]) contains a note that gives a hint why this choice was made:

> Without compromising the security proof for RSASSA-PSS, one may perform steps 1 and 2 of EMSA-PSS-ENCODE and EMSA-PSS-VERIFY (the application of the hash function to the message) outside the module that computes the rest of the signature operation, so that mHash rather than the message M itself is input to the module.

The design of RSASSA-PSS allows to separate the first hashing step from the rest of the signature operation. This is especially important for devices like smart cards where data transfer is limited. One can design applications that only calculate the hash of a message and transfer that to a cryptographic unit (like a smart card) that does the signature operation. Beside that, existing signature software implementations often rely on the separation of message hashing and the signature operation, because pretty much all signature schemes until now start with a hashing operation.

So there is a trade-off between a more secure design (PSS96) and less obstacles in certain implementation scenarios. In the standardization process, the less secure design was chosen. Due to this design decision, the combination of RSASSA-PSS and a hash function that doesn't provide collision-resistance creates an insecure algorithm. So it is crucial that a secure hash function is taken. This is especially important to note as the existing standards for RSASSA-PSS define SHA-1 as the default hash function, which can not be considered collision-resistant any more.

5.3 Randomized Hashing

In the aftermath of the collision attacks against MD5 and SHA-1, Shai Halevi and Hugo Krawczyk proposed a method called randomized hashing at the Crypto 2006 conference [Halevi and Krawczyk, 2007] . Later, the NIST has adopted this in SP-800-106 [Dang, 2009] . Currently there are no efforts underway in standardizing randomized hashing in cryptographic protocols.

The idea of randomized hashing is this: Before the generation of a hash, a random value rv of fixed size should be generated. Then, by repeating rv, a string Rv is generated with the same size as the message input m (this is basically an XOR-Vigenère cipher of M with rv). Finally, the input to the hash function is:

$M = rv||(m \oplus Rv)||rv\_length\_indicator$

rv has to be shipped along with the signature to allow verification. This is a problem, as this would often imply substantial changes to allow the use of randomized hashing within existing protocols. Randomized hashing has the effect that even if the underlying hash function is not collision resistant, a signature scheme based on randomized hashing stays secure. Halevi and Krawvzyk call this property eTCR (enhanced Target Collision Resistance).

The NIST document explicitly suggests this scheme for RSASSA-PSS and states that the salt value from PSS could be used as rv.

Now, the interesting thing is that randomized hashing has a very similar effect as the first salting and hashing step in PSS96, the original variant of PSS. Due to the input randomization, PSS96 also provides eTCR. In fact, RSASSA-PSS with randomized hashing looks very similar to PSS96, the only difference is another (unnecessary) hashing step and the XOR-ing of the message. So by applying randomized hashing on RSASSA-PSS, we get back the security we lost with the changes from PSS96.

The additional XOR-ing adds additional security, but only for very rare scenarios. PSS96's first step is $Hash(salt|M)$ – an attacker able to create two messages M_1 and M_2, which create a collision for an unknown prefix would be able to create a collision here. Although it is unlikely that such an attack scenario exists, randomized hashing would prevent even that.

Dan Boneh and Weidong Shao provide an implementation of randomized hashing for nss [Boneh and Shao, 2007] . With that, they also provide a proposal how to use this within X.509 certificates, however, it seems that currently there is no effort to push this into an official IETF standard. They use an X.509 certificate extension to ship the random value rv with the certificate. This makes their scheme to be only usable within X.509 and there is no easy way to adopt this to other protocols (for example CMS).

The combination RSASSA-PSS and randomized hashing would avoid the need to transfer rv separately if one sets $salt = rv$, as it is suggested by the NIST.

5.4 Randomization in SHA-3 Candidates

As already mentioned, currently a contest for the new cryptographic hash standard SHA-3 is going on. At the moment there are five candidates left in the competition: BLAKE, Grøstl, JH, Keccak and Skein. The idea of randomized hashing influenced several of them.

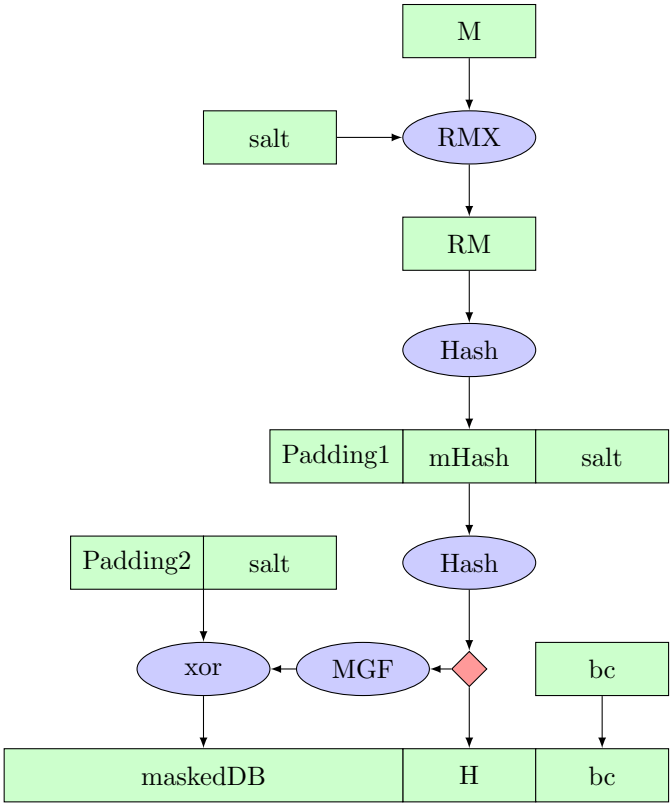


Figure 5: RSASSA-PSS in combination with randomized hashing

The BLAKE hash function has two inputs: The message and an optional salt. The BLAKE paper explicitly mentions that this can be used for randomized hashing (chapter 4.4, page 28 in [Aumasson et al., 2010]) .

In the proposal for the Grøstl hash function, randomized hashing is also mentioned (chapter 6.2, page 18 in [Gauravaram et al., 2011]) . However, Grøstl has no dedicated mode for randomized hashing, the authors suggest that the method proposed by Halevi/Krawczyk and the NIST can be used without modification.

The Keccak authors propose "keyed or randomized modes simply by prepending a key or salt to the input message" [11]. This is roughly the same that is done in PSS96.

The most interesting concept is in Skein. They define the possibility to use several optional inputs to the hash function (chapter 2.5, page 6 in [Schneier et al., 2010]) . Like BLAKE, Skein offers the possibility to inject a salt (they call it nonce, but that is the same), making the direct use of Skein as a randomized hashing function possible. The Skein paper proposes further to use a unique identifier for every protocol as an additional input to avoid interaction between protocols [Schneier et al., 1997] . It also suggests to use the public key in a cryptographic operation as an additional input. The definition of further optional arguments is possible and recommended.

5.5 Summary

We have seen that in the standardization process of PSS, a questionable tradeoff has been made between security and easier implementation. Through direct randomization of the input with a salt the original PSS96 proposal provides enhanced target collision resistance (eTCR).

However, the idea of randomized hashing and the ongoing SHA-3 competition provide an opportunity to get this eTCR property back within upcoming standards.

6 Considerations for Implementations

6.1 Hash Algorithm

PSS requires a hash function and a mask generation function derived from a hash function. While it is in theory possible to use two different hash functions for that, it is suggested to use the same for both for simplicity. Still, a combination of two secure hash functions (for example SHA-256 as the input hash and MGF1 with SHA-512 as the mask generation function) does not cause any security problems.

[11] keccak website at `http://keccak.noekeon.org/`

PKCS #1 v2.1 specifies the use of PSS with either SHA-1 or one of the SHA-2 (SHA-256, SHA-384, SHA-512) algorithms. Later standards like RFC 4055 (PSS signatures for X.509) additionally allow SHA-224. The proposed default is SHA-1.

However, it is highly suggested not to use SHA-1 anymore. We already discussed the weaknesses of still widely used hash functions MD5 and SHA-1 in chapter 2.5. Up until now, the SHA-2-family (SHA-224, SHA-256, SHA-385, SHA-512) can be considered secure and is the only widely accepted hash standard without known weaknesses.

As PSS is a safety measure against unknown attacks, it barely makes any sense to use it with SHA-1, where attacks are already known. The switch from MD5/SHA-1 to SHA-2 should have a higher priority than the implementation of PSS padding.

6.2 Key Size

For a long time, RSA key sizes between 512 and 768 bit were quite common. They should be considered completely insecure today, although they still can be found in real-world applications. In 2010, a research team factorized a 768 bit number [Kleinjung et al., 2010] . In 2009, it was possible to factorize a 512 bit key used for signing the operating system of a Texas Instruments calculator by a private person on a home computer [ticalc.org, 2009] . Those attacks have also questioned the security of RSA with 1024 bit.

In 2003, Adi Shamir (one of the original RSA authors) proposed a theoretical design for a hypothetical device called TWIRL (The Weizmann Institute Relation Locator) that would be capable of factoring large numbers up to 1024 bit in less than a year with a 10 million dollar device [Shamir and Tromer, 2003] . This would fully break RSA with a key size of 1024 bit or lower, meaning that the private key can be revealed with knowledge of the public key.

Due to the high costs, nobody has publicly built a TWIRL device. However, as with the transition to new hash functions, the transition to key sizes above 1024 bit should have higher priority than the transition to PSS and while in theory possible, it barely makes sense to use PSS with a RSA key size of 1024 bit. RSA Inc. as well as the NIST propose not to use 1024 bit keys after 2010 any more.

6.3 Exponent

In its early days, RSA was done with a random, large-sized exponent. Later, it became prevalent that for better performance in the encryption / verification process, very small exponents like $e = 3$ could be used. Textbook RSA in combination with a small exponent raises a number of issues, but all of them can be avoided with padding and hashing.

However, we already saw that the Bleichenbacher attack was only possible with a very small exponent like $e = 3$. So it seems small exponents are not a problem per se, but a risky choice regarding implementation problems.

Usually today's RSA implementations use an exponent of $e = 65537$ - a tradeoff between very big exponents that make verification very slow and very small exponents that seem risky. $e = 65537$ avoids all known attacks against small exponents. The NIST recommendations do not allow exponents smaller than 65537 (page 6 in [NIST, 2010]) .

6.4 Separating Keys for different Schemes

We saw in chapter 2.4 that when using plain RSA, it causes severe problems when using the same key for signature generation and encryption. While with any kind of padded / hashed RSA this is not a direct issue, it is still considered bad practice to use the same key for different purposes. This is not limited to RSA, it is always a useful safety measurement to use key material only for one purpose due to unconsidered protocol / algorithm interactions. Schneier, Wagner and Kelsey have investigated this [Schneier et al., 1997] and have come to the conclusion that it is advisable to fixate the purpose of key material directly with the key generation.

For our case, this would mean that in an ideal case, an RSA key (e.g., in an X.509 certificate) would be limited to either RSASSA-PSS or RSAES-OAEP. Neither should one key be used for both PSS and OAEP, nor should the same key be used for old (PKCS #1 1.5) and new (PSS) padding.

6.5 Summary

When implementing RSA with PSS, we also need to make a couple of other design decisions. For best security properties, it is advisable to use a secure hash function without collision problems (currently one of the SHA-2 family ones, in the future probably SHA-3), a decent key size (2048 bit or more) and an exponent not to small (65537 or above).

It also looks like a good idea to limit the key usage exclusively to PSS and not to use the same keys for different padding schemes.

7 Protocols – Standards and Implementations

In this chapter, we will have a look at some common cryptographic high level protocols and their support of RSASSA-PSS.

Engine	Default / no parameters	With parameter block	Differing hashes	PSS-keys
OpenSSL latest/1.0.0d [a]	✗	✗	✗	✗
OpenSSL CVS/1.1	✓	✓	✓	✗
nss latest/3.12.9 [b]	✗	✗	✗	✗
nss CVS+patches	✓	✓	✓	✗
GnuTLS latest/2.12.2	✗	✗	✗	✗
Windows Vista/7 SChannel.dll [c]	✓	✓	✗	✗
MacOS 10.6.7 [d]	✓	✗	✗	✗
IAIK java library	✓	✓	✓	✓
BouncyCastle java library	✓	✓	✓	✗

[a] used by Opera
[b] used by Firefox, Thunderbird, Chromium/Chrome on Linux
[c] used by Internet Explorer, Chromium/Chrome on Windows, Safari on Windows
[d] used by Safari on MacOS

Figure 6: PSS support in X.509 implementations

7.1 X.509

X.509 is a standard to provide certificates which can be used to ship public keys for further cryptographic operations. Sometimes X.509 certificates are called SSL certificates, but this is not accurate – SSL / TLS is their most common usage, but X.509 certificates are more generic and can be used in a large variety of protocols.

In RFC 4055 [IETF Network Working Group, 2005a] , the use of the PKCS #1 v2.1 primitives within X.509 certificates and certificate revocation lists is specified, RFC 5756 [IETF Network Working Group, 2010] contains some minor updates. X.509 certificates are used in a wide variety of applications, their most common use is in combination with SSL / TLS.

RFC 4055 allows two things: Generating signatures with RSASSA-PSS on other X.509 certificates and creating keys designated for RSASSA-PSS and RSAES-OAEP. Signatures can also be generated with "normal"/old RSA keys without a designated use case. However, designated RSASSA-PSS keys are barely supported anywhere at all.

Implementations of RFC 4055 have been lacking until recently. Most applications are based on four SSL / X.509-engines. The Microsoft Windows and Apple MacOS X operating systems bring their own cryptographic engine that is used by their own browsers (Internet Explorer, Safari) and partly by others

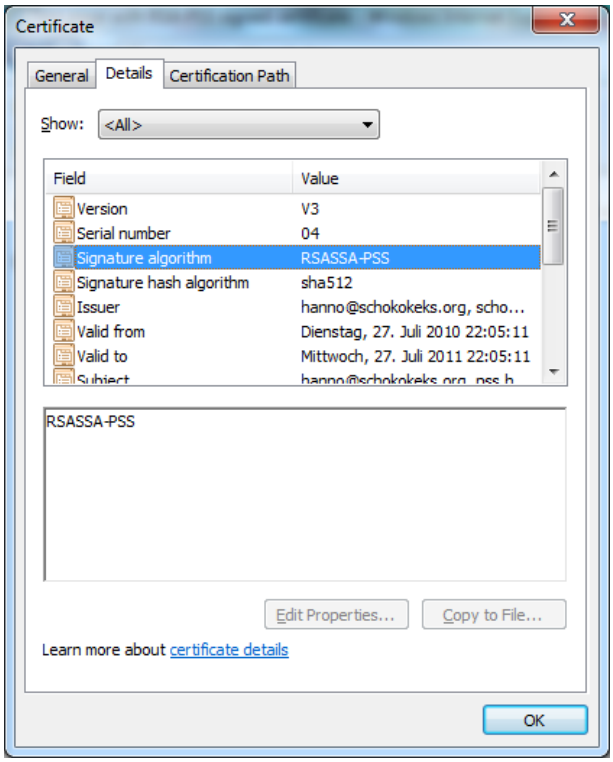


Figure 7: RSASSA-PSS certificate with Internet Explorer on Windows 7

(Chrome/Chromium). There are two widely used free software implementations of X.509 and SSL: nss (Network Security Service, used by Mozilla Firefox/Thunderbird and Google Chrome/Chromium on Linux) and OpenSSL (used by the Apache web server, the Opera browser and many others). Slighter less used is GnuTLS.

I myself have contributed code for RSASSA-PSS support to the nss / Mozilla project during the Google Summer of Code 2010 (we will cover implementation details later in chapter 8). It will probably be part of one of the next major Firefox updates (5.0 or 6.0).

PSS support is also present in the current CVS code of OpenSSL and will be part of the upcoming 1.1 release. With the `openssl` command line tool, it can be enabled with the parameter `-sigopt rsa_padding_mode:pss` and the salt length can be set with `-sigopt rsa_pss_saltlen:64`. By default, `openssl` makes the salt as long as possible.

Microsoft Windows supports PSS signatures since Windows Vista, but fails with some uncommon parameters. It does not support SHA224 as the hash function (this is not limited to PSS) and it fails when the input hash function is different from the mask generation function's hash function.

MacOS X supports only PSS signatures with default parameters. This limits the support to signatures with the weak hash function SHA1.

There also exist two Java libraries which support RSASSA-PSS signatures. The free software project Bouncy Castle supports certificate signatures.

The most complete RSASSA-PSS support is in a Java library by the Institute for Applied Information Processing and Communication (IAIK) at the Technical University of Graz. It is the only implementation I'm aware of that is able to generate designated RSASSA-PSS certificates. A free version for educational and research purpose can be downloaded at their website[12].

The US-based Electronic Frontier Foundation runs the SSL Observatory[13], a research project that collects all certificates on https connections in the full public IPv4 space. The complete database contains about 12 million certificates. Out of them, only four certificates on two IPs listed that are signed with RSASSA-PSS. They were not signed by any browser-accepted certificate authority.

7.2 Cryptographic Message Syntax (CMS) and S/MIME

The Cryptographic Message Syntax (CMS), based on the older PKCS #7, specifies encryption and signatures for generic messages. Its main use is within S/MIME, a widely used standard for email encryption and signing. It is based on X.509 certificates.

RSASSA-PSS and RSA-OAEP for CMS have been specified within RFC 4056 [IETF Network Working Group, 2005b] . CMS signatures are very similar to signatures within X.509.

I am not aware of any implementation of RSASSA-PSS for Cryptographic Message Syntax.

7.3 PKCS #11

PKCS #11 is a generic abstraction API defining cryptographic operations and objects. PKCS #11 ships header files defining constants for cryptographic objects such as keys, signatures and algorithms. A constant starting with `CKM_` describes a "method", basically meaning an algorithm. Since PKCS #11 version 2.11, it knows the method `CKM_RSA_PKCS_PSS` (chapter 12.1.9, page 201 in [RSA Inc., 2004]) .

PKCS #11 also has an abstraction for the PSS parameter block named `CK_RSA_PKCS_PSS_PARAMS`. This defines a C struct containing all the relevant meta information for a signature. There is a slight conceptual difference from the ASN.1 structs from PKCS #1: Every combination of mask generation function

[12] `http://jce.iaik.tugraz.at/`
[13] `https://www.eff.org/observatory`

and hash algorithm gets its own constant, e.g., `CKG_MGF1_SHA256`, while in PKCS #1 there is an additional parameter to the mask generation function for the hash function.

PKCS #11, however, does not have a way to define designated PSS keys. PKCS #11 2.20 only has one key type for RSA – `CKK_RSA` – it does not differentiate the use of the RSA key.

7.4 IPsec

IPsec is a protocol to add a cryptographic layer on the IP protocol. IPsec is composed by a large number of sub-protocols. In the IETF-standard RFC 4359 [IETF Network Working Group, 2006] , the use of RSASSA-PSS in the Encapsulating Security Payload (ESP) and Authentication Header (AH) protocols part of the IPsec protocol is specified. However, it is not possible to use any other hash function than SHA-1 (see chapter 6.1).

7.5 XMLDSig, XMLenc

For XML documents, the W3C has two cryptographic specifications, XMLenc for encryption [W3C, 2002] and XMLDSig for signatures [W3C, 2008] . While XMLenc supports OAEP encryption, PSS support in XMLDSig is lacking. A proposal from 2007 [Lanz et al., 2007] exists but has not been adopted in the latest revision of the standard in 2008. Due to the known weaknesses of SHA-1, this proposal uses SHA-256 as the default hash function.

Unlike X.509, XMLDSig does not use Object Identifiers for algorithms, instead it uses an URL-based scheme. The URL directly refers to the algorithm specification on the W3C web page.

The proposal of Lanz defines these identifiers:
`http://www.example.org/xmldsig-pss/#rsa-pss`
`http://www.example.org/xmldsig-pss/#mgf1`

Once standardized, `http://www.example.org/` will be replaced by something like `http://www.w3.org/2011/09/xmldsig-pss`.

7.6 No Support yet: OpenPGP, DNSSEC, TLS

The software Pretty Good Privacy (PGP) by Phil Zimmerman made email encryption popular in 1991. Today, PGP and other implementations like the free software GPG are based on the OpenPGP standard RFC 4880 [IETF Network Working Group, 2007] . Although PKCS #1 v2.1 had already been released when the standard was written, it exclusively uses EMSA-PKCS1-v1_5 and there are no plans to change that yet. RFC 4880 also lists a couple of reserved algorithm IDs, e.g., for elliptic curve cryptography, but none are listed for OAEP/PSS.

DNSSEC is a security extension to the domain name system. It was developed back in 1999, but saw no widespread use for quite a long time. In 2008, security researcher Dan Kaminsky presented a real-world attack on the caching of DNS [US-CERT, 2008] . Through mitigation measures, it was possible to make such attacks much harder, still DNSSEC is considered the only long-term solution for a reliable DNS. All domain name authorities are working on implementing DNSSEC and the root zone has been signed in 2010. The latest signature algorithms based on RSA and SHA-2 are specified in RFC 5702 and use EMSA-PKCS1-v1_5 (section 3 in [IETF Network Working Group, 2009]) . Section 8.1 contains a note that this has been decided to make the transition from the SHA-1 algorithms easier. It is unlikely that this will change, as the current plans of the DNS working group at the IETF are to switch to elliptic curve algorithms in the future.

DKIM (DomainKeys Identified Mail) is a signature system for outgoing emails to prevent spam. Similar to DNSSEC, it was developed long after the standardization of PSS, but supports exclusively RSA-PKCS1-v1_5, because its authors feared it would make implementation too difficult if they required a scheme not widely supported.

The Transport Layer Security protocol (formerly SSL) is widely used to secure existing protocols, like https, pop3s, smtps etc. The latest version does not support any padding beside RSASSA-PKCS1-v1_5.

As shown in this chapter, a couple of the most significant cryptographic protocols don't support RSASSA-PSS at all and there are no transition plans.

7.7 Other Protocols using PSS

A small number of less widely used protocols implement PSS.

Microsoft's digital rights management system COPP is using RSASSA-PSS signatures to verify signatures on graphics drivers. However, it uses SHA-1 as a hash algorithm and sets the salt length to zero [Microsoft, 2010] .

The European standard for smart cards prEN 14890-1:2008 defines an interface for RSASSA-PSS based signatures (part 2, page 14, chapter 6.3.2 in [Technical Committee CEN/TC 224, 2008]) . It states that the hash for the mask generation function and the input have to be the same.

7.8 Summary

We have seen that the support for PSS within cryptographic protocols is very limited. Many important protocol standards don't support it at all and from the ones that do, implementations are rare.

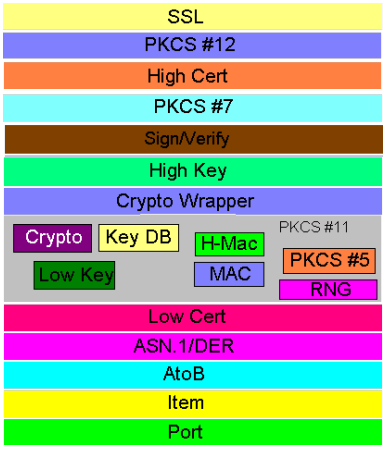


Figure 8: nss layers visualized, from `http://www.mozilla.org/projects/security/pki/nss/nss-guidelines.html`

8 Implementation of X.509 PSS Signatures in nss

The security library nss (Network Security Service) is the cryptography backend for the Mozilla Firefox browser and Thunderbird email client. It originates from the SSL library in the Netscape browser when Netscape originally invented SSL.

Today, nss is published under a free software license (GPL, LGPL or MPL). Beside the Mozilla products, it is also used by Google's Chrome/Chromium browser on Linux and it is possible to use it in the Apache web server with mod_nss.

Within the Google Summer of Code 2010, I implemented support for RSASSA-PSS signatures according to RFC 4055 and RFC 3447 in nss. It is however not yet completely included in the nss CVS source.

All necessary patches for PSS support and a patched version of the nss code are provided here:

`http://rsapss.hboeck.de/download/current/`

I will keep them updated until the code is fully included in nss.

8.1 nss Library Layers

The nss library source code consists of several layers. They are visualized in figure 8. Usually everything above the PKCS #11 layer is stacked, meaning every layer's functions can only be accessed by the upper layer. The layers below the PKCS #11 layer are helper functions available from everywhere in the code.

```
static SECStatus
emsa_pss_verify(const unsigned char *mHash,
                const unsigned char *em, unsigned int emLen,
                unsigned int emBits, HASH_HashType hashAlg,
                HASH_HashType maskHashAlg, unsigned int sLen)
{
    const SECHashObject *hash;
    void *hashContext;
    unsigned char *db;
    unsigned char *H_;  /* H' from the RFC */
    unsigned int i, dbMaskLen, zeroBits;
    SECStatus rv;

    hash = HASH_GetRawHashObject(hashAlg);
    dbMaskLen = emLen - hash->length - 1;

    /* Step 3 + 4 */
    if ((emLen < (hash->length + sLen + 2)) ||
        (em[emLen - 1] != 0xbc)) {
        PORT_SetError(SEC_ERROR_BAD_SIGNATURE);
        return SECFailure;
    }

    /* Step 6 */
    zeroBits = emLen*8 - emBits;
    if ((em[0] >> (8 - zeroBits)) != 0) {
        PORT_SetError(SEC_ERROR_BAD_SIGNATURE);
        return SECFailure;
    }

    /* Step 7 */
    db = (unsigned char *)PORT_Alloc(dbMaskLen);
    if (db == NULL) {
        PORT_SetError(SEC_ERROR_NO_MEMORY);
        return SECFailure;
    }
    /* &em[dbMaskLen] points to H, used as mgfSeed */
    MGF1(maskHashAlg, db, dbMaskLen, &em[dbMaskLen], hash->length);

    /* Step 8 */
    for (i = 0; i < dbMaskLen; i++) {
        db[i] ^= em[i];
    }
```

Listing 3: Beginning of emsa_pss_verify from rsawrapr.c

```
/* CK_RSA_PKCS_PSS_PARAMS is new for v2.11.
 * CK_RSA_PKCS_PSS_PARAMS provides the parameters to the
 * CKM_RSA_PKCS_PSS mechanism(s). */
typedef struct CK_RSA_PKCS_PSS_PARAMS {
        CK_MECHANISM_TYPE    hashAlg;
        CK_RSA_PKCS_MGF_TYPE mgf;
        CK_ULONG             sLen;
} CK_RSA_PKCS_PSS_PARAMS;
```

Listing 4: PSS parameter struct from PKCS #11 2.2 / pkcs11t.h

8.2 Object Identifiers

The first trivial task was letting nss know the Object Identifiers. nss has numeric constants for every OID it knows, which are listed in `util/secoidt.h` and mapped to OIDs and descriptions in `util/secoid.c`. Relevant for PSS are `SEC_OID_PKCS1_RSA_PSS_SIGNATURE` and `SEC_OID_PKCS1_MGF1` (I also added the OIDs required for OAEP, although they are not used yet).

8.3 freebl, MGF1

The basic cryptographic operations are within the `freebl` directory (bl is an abbreviation for bottom layer). For the RSA part, no changes were required, as the RSA primitive itself does not change with PSS. But PSS requires a mask generation function – specifically MGF1 – which is a new cryptographic primitive. So one of the first steps was adding a primitive for MGF1 to `freebl/mgf1.c`.

8.4 PSS Padding and Verification Code

The actual padding for RSA happens within the `softoken` layer. For PSS, I added two new public functions `RSA_CheckSignPSS` and `RSA_SignPSS` (analogous to `RSA_CheckSign` and `RSA_Sign`, which are for old PKCS #1 v1.5 padding). They call (non-public) functions emsa_pss_verify / emsa_pss_encode, which are implementations of the pseudo-functions EMSA-PSS-VERIFY / EMSA-PSS-ENCODE from PKCS #1 v2.1 (section 9.1.1 and 9.1.2 in [RSA Inc., 2002]) . However, instead of using the message as the input, we take the hash of the message as the input, which is explicitly allowed within the standard (section 9.1, point 3 in [RSA Inc., 2002]) .

8.5 PKCS #11 Module

In nss, all access from higher level functions to cryptographic operations works through an implementation of PKCS #11. PKCS #11 defines a C struct `CK_RSA_PKCS_PSS_PARAMS` that contains the same information as the ASN.1 parameter block used in X.509. I made use of this and store the information in

such a struct to pass it down through all API layers.

The PKCS #11 layer causes a major problem regarding PSS-only keys. RFC 4055 allows not only using RSASSA-PSS as the signature algorithm, but it also allows the key within a certificate to be limited exclusively to PSS. It even allows specifying a defined set of parameters allowed for this key. From a cryptographic point of view, this is generally a good idea. However, as mentioned in chapter 7.3, PKCS #11 2.20 only knows one key type for RSA (`CKK_RSA`) and has no way to specify a designated PSS key. For this reason, I saw no easy way of implementing such key types within nss in the given framework, although in the long term it is strongly recommended to use such restricted keys. This should be considered in the next review of the PKCS #11 standard. I have sent a comment to the PKCS feedback mail address on this, but got no reply.

8.6 Upper Layers

The remaining changes needed were in the `cryptohi`-layer. It provides high-level API functions for signature verification. Here, I had to implement routines for decoding and encoding the PSS parameter block (`PSSU_DecodeDERPSSParams` and `PSSU_EncodeDERPSSParams`), for which i added a new file `cryptohi/pssutil.c`. They convert the parameter block from ASN.1 to the PKCS #11 struct `CK_RSA_PKCS_PSS_PARAMS` mentioned above and back. They get called while decoding the signature algorithm from `sec_DecodeSigAlg` in `secvfy.c`.

Further work included mostly the already mentioned parameter passing through all functions and some new public API functions that allow signature creation with parameters. For example, analogous to the function `SEC_DerSignData`, I added a new function `SEC_DerSignDataWithParams`.

8.7 Tools and Frontends

nss ships a couple of command line tools to access the functionality in the libraries. Certificate management can be done with `certutil`. I added an additional parameter `--pss`, which allows the creation of RSASSA-PSS signatures with any parameter set. Although this hardly makes any sense, I even allowed to set a combination of one hash function for the input hash and another one for the mask generation function. As it is part of the specification, this can at least be used to test other implementations.

The following commands would create a sample, self-signed PSS certificate and export it:

```
certutil -N -d [dir]
certutil -S -2 -v 9999 -g 2048 -Z SHA512 --pss MGF1:SHA512:32 -n
mycert -s "cn=mycert" -x -t "C" -d [dir]
certutil -L -d [dir] -n mycert -a
```

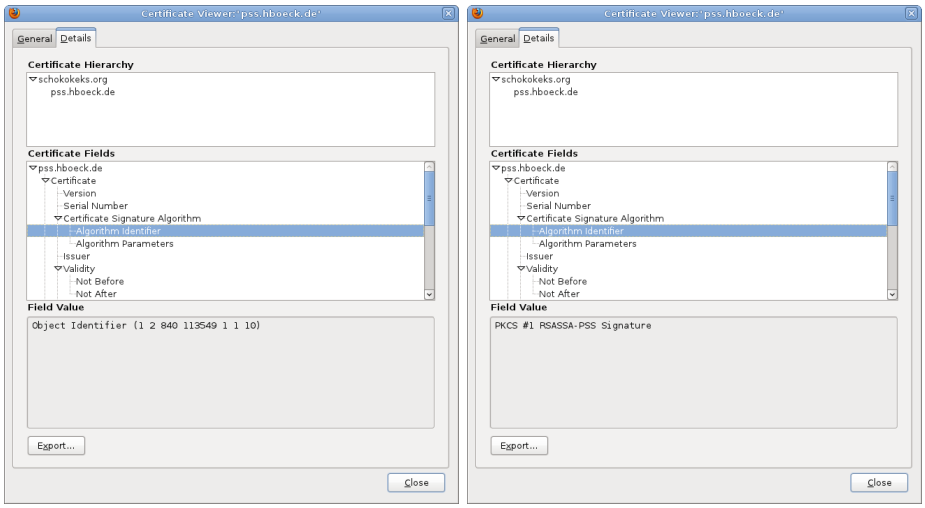


Figure 9: Certificate signed with RSASSA-PSS in Firefox, old and new.

`[dir]` must be replaced with a directory name where the certificate database will be generated and stored. The `--pss` parameter can be followed by a combination of mask generation function, appertaining hash function and salt length. It is also possible to specify `--pss default` - this will select the defaults from the standard (MGF1 with SHA-1 and a salt length of 32 Bytes). The hash function for the input message is not specific to PSS and can be specified through the `-Z` parameter.

8.8 Firefox

When the nss library is able to verify RSASSA-PSS signatures on certificates, this is also true for any application using that library. Therefore, Firefox itself needs no code changes to be able to verify PSS certificates. The only cosmetic change that is necessary is to let Firefox know about the Object Identifier, so the GUI can correctly display the used certificate algorithm (see Figure 9).

8.9 Further work

During my work on nss, I tested my code with the debugging tool Valgrind for memory leaks, which helped a lot to spot bugs in my own work. The existing command line tools from nss had a couple of own memory leaks I eliminated during that work to make the checking easier[14].

[14] `https://bugzilla.mozilla.org/582800` and `https://bugzilla.mozilla.org/581804`

8.10 Difficulties

I found out that the most difficult part was not the implementation of the PSS padding itself, but the handling of the parameter block. RFC 3447 specifies a parameter block (s. 49 in [RSA Inc., 2002]) allowing to specify the hash function, salt length, mask generation function and the hash function used inside the mask generation function. The parameters are decoded (cryptohi/secvfy.c) several API layers above the signature generation (softoken/rsawrapr.c) and have to be passed through them (pk11wrap/pk11obj.c). None of the existing algorithms within nss had a parameter block, so this required a bunch of API changes.

8.11 Conclusions from the Implementation

The parameter block adds a lot of complexity for no real gain. It allows for example constructions where the hash function used directly and the hash function for the mask generation function differ (e.g., SHA-1 with MGF1 using SHA-256). The variable salt length allows to choose values that violate the security properties of the whole PSS construction (e.g., a zero length salt). It would've been much easier to stick to a predefined set of sane parameters and assign them their own IDs.

Another point where unnecessary complexity was added was the handling of key sizes. A number of bit-shifting operations were only necessary because RFC 3447 allows keys of any size. It would have made the whole implementation significantly simpler if the allowed key sizes were restricted to ones divisible by 8 (which makes a byte-representation without padding possible). As usually all real-world applications use common key sizes like 1024, 2048 or 4096, this restriction would not have many implications on users.

The fact that the only input to the whole signature generation and verification functions is the hash of a message was welcome for the implementation, but it weakens the security (see chapter 5). It would've needed even more API changes to make a construction like the original PSS proposal that does not start with $Hash(M)$ possible. However, I would still consider randomization of the hash a good idea, as it would vastly improve the security on real-world attacks.

8.12 Summary

I have provided working patches to support RSASSA-PSS in nss, a cryptographic library widely used by popular products like the Mozilla Firefox browser. Due to the very invasive changes required, inclusion of them into the main nss codebase will probably still require some time.

Further possible tasks that could be done are support for designated RSASSA-PSS keys/certificates and support for Cryptographic Message Syntax

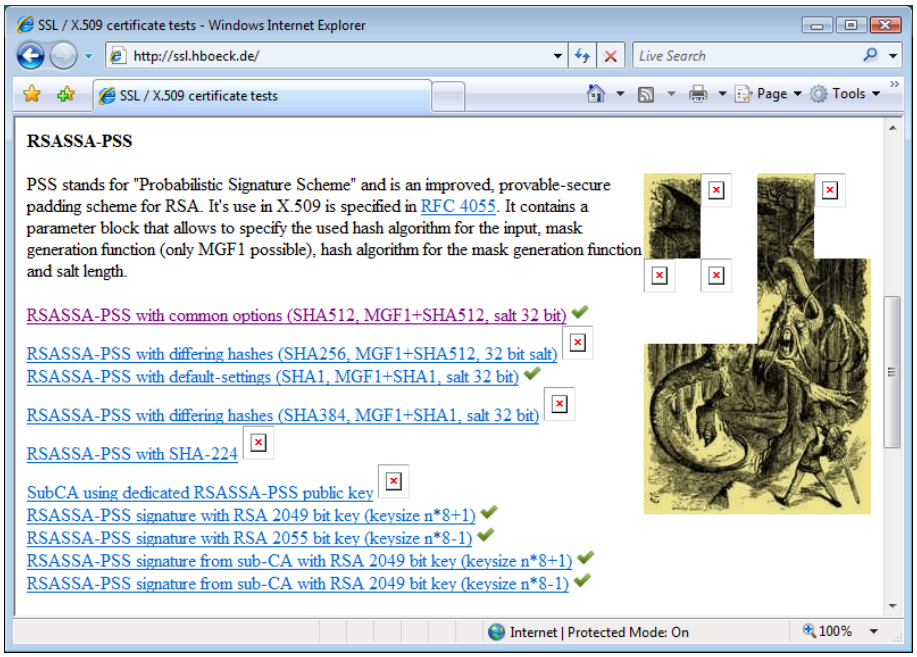


Figure 10: Online test in Internet Explorer / Windows Vista

and S/MIME email encryption.

9 Online Tests with X.509 Certificates

I have set up a number of test web hosts with X.509 certificates signed with RSASSA-PSS. General information can be found here:
`http://ssl.hboeck.de/`

Various certificate samples (not only PSS related) are set up on subdomains. There is no browser-accepted certificate authority that will issue certificates with uncommon algorithms, so all certificates are just signed by my own testing certificate authority. But its root certificate can easily be installed in most browsers:
`http://ssl.hboeck.de/rootca.crt`

A sample with an RSASSA-PSS signature with common settings can be found at
`https://pss.ssl.hboeck.de/`

Various uncommon combinations of algorithms can also be found, for example using a different hash for the message input (SHA-256) and the mask generation function (SHA-384):
`https://pss-sha256-mgf1-sha512-20.ssl.hboeck.de/`

During the implementation of RSASSA-PSS in nss, I found out that the handling of uncommon key sizes are especially prone to errors. I also discovered a bug in OpenSSL with PSS certificates using a key size which is a multiple of 8 plus 1[15], which has been fixed fast by the OpenSSL developers. To check for such issues, I created test cases with intermediate certificates with a key size of 2049 (n*8+1) and 2055 bit (n*8-1):
`https://pss-subcert-2049.ssl.hboeck.de/`
`https://pss-subcert-2055.ssl.hboeck.de/`

I also created an intermediate certificate with a key bound to RSASSA-PSS, however I'm not aware of any browser capable of verifying that at the moment:
`https://pss-subca.ssl.hboeck.de/`

The test page includes images with green ticks included from all the subdomains, so one can easily see what is supported at a glance. Also, a bigger image that is composed out of parts fetched from the different subdomains is displayed (see screenshot at figure 10) - in an ideal case with all options supported, the image is displayed complete.

Due to my lack of owning many IP addresses, all those hosts/certificates can only be accessed either through IPv6 or through Server Name Indication (SNI), a TLS extension that allows more than one X.509 certificate on one IP. While all common browsers (Internet Explorer, Firefox, Safari, Chrome, Opera) support SNI, some special purpose browsers still do not (for example the Android default browser).

10 Public Authorities, Research and Industry Organizations

A number of public authorities, research and industry organizations provide recommendations and regulations for cryptographic algorithms. If their suggestions are required by some law, this can be an especially important push to deploy better cryptography.

10.1 Electronic Signatures in the EU

The European Union has a framework for digital signatures. Directive 1999/93/EC [EU, 1999] defines rules for different kinds of electronic signatures that should be interchangeable through the European Union. The directive itself has no specific requirements on cryptographic algorithms, it is up to every country to define a set of suitable algorithms. But there is a list of recommendations by the European Telecommunications Standards Institute (ETSI). Many countries just refer to the ETSI document in their own signature laws.

The latest ETSI recommendations from 2007 (chapter 7.2, page 23 in

[15] http://rt.openssl.org/Ticket/Display.html?id=2315

[ETSI, 2007]) consider RSASSA-PSS to be better especially for long term use. For applications requiring signature validity for up to 10 years, PSS with a minimum salt length of 64 bit is the only suggested padding scheme (chapter 9.3, page 29 in [ETSI, 2007]) .

10.2 Electronic Signatures in Germany

Germany, unlike most other EU-countries, produces its own list of cryptographic algorithms allowed according to its signature law. The Bundesnetzagentur produces a yearly document commonly called "Algorithmenkatalog". It is in various aspects stricter than the ETSI recommendations. It says that signatures with RSASSA-PKCS1-v1_5 may not be used after 2014, certificate signatures with RSASSA-PKCS1-v1_5 may not be issued after 2016 (chapter 3.1, page 6 in [Bundesnetzagentur, 2011]) . It is suggested not to use RSASSA-PKCS1-v1_5 after 2013 anymore.

Other documents by German authorities reference the "Algorithmenkatalog". For example, the technical policy TR-03125 by the BSI (Bundesamt für Sicherheit in der Informationstechnik) for long-term archiving of documents says that the recommendations from the current "Algorithmenkatalog" apply for signature algorithms (chapter 4.3, page 10 in [BSI, 2011]) . However, signatures within this technical policy are XML signatures – which, as we saw earlier, don't support PSS yet (see chapter 7.5). So these policies might likely contradict each other if RSASSA-PSS within XMLDSig is not standardized soon.

10.3 Electronic Passports

While https and SSL / TLS in general are the most common use cases of X.509, they are not the only one. Most international passports today contain signatures on a chip from a so-called Country Signing Certificate Authorities (CSCA). In a technical report by the International Civil Aviation Organization (ICAO), the recommendation is to use RSASSA-PSS and states are required to be able to verify both RSASSA-PSS and RSASSA-PKCS1_v15 (chapter 8.2 in [ICAO, 2006]) .

The ICAO provides a worldwide list of CSCA root certificates at
`https://pkddownloadsg.icao.int`

It contains 192 certificates from Japan, South Korea and Canada that are signed with RSASSA-PSS. The majority (3504) uses RSASSA-PKCS1_v15 with SHA-256, 142 use RSASSA-PKCS1_v15 with SHA-1. 36 use ECDSA with SHA-1, 18 use ECDSA with SHA-256.

```
Certificate:
 Data:
  Version: 3 (0x2)
  Serial Number: 27 (0x1b)
  Signature Algorithm: PKCS #1 RSA-PSS Signature
   Parameters:
    Hash algorithm: SHA-256
    Mask algorithm: PKCS #1 MGF1 Mask Generation Function
    Mask hash algorithm: SHA-256
    Salt Length: 32 (0x20)
  Issuer: "CN=e-passportCSCA,OU=The Ministry of Foreign
   Affairs,O=Japanese Government,C=JP"
  Validity:
   Not Before: Thu Aug 30 01:58:47 2007
   Not After : Thu Aug 30 01:58:47 2018
  Subject: "CN=Minister for Foreign Affairs,OU=e-passportDS,
   OU=The Ministry of Foreign Affairs,O=Japanese Government,C=JP"
  Subject Public Key Info:
   Public Key Algorithm: PKCS #1 RSA Encryption
   RSA Public Key:
    Modulus:
     d7:e3:e2:df:f0:d5:1e:f0:1f:0c:88:cd:1a:0c:7d:5b:
     [...]
    Exponent: 3 (0x3)
  Signed Extensions:
   Name: Certificate Authority Key Identifier
   Key ID:
    62:5b:86:8c:78:da:3e:31:95:e8:39:22:fe:75:2d:40:
    75:ce:a0:90

   Name: Certificate Key Usage
   Critical: True
   Usages: Digital Signature

   Name: Certificate Policies
   Data:
    Policy Name: OID.1.2.392.100350.6.5.1.1.2

 Signature Algorithm: PKCS #1 RSA-PSS Signature
  Parameters:
   Hash algorithm: SHA-256
   Mask algorithm: PKCS #1 MGF1 Mask Generation Function
   Mask hash algorithm: SHA-256
   Salt Length: 32 (0x20)
 Signature:
  0b:eb:bb:67:83:af:58:cf:a0:2c:f9:1e:96:71:a7:68:
  [...]
 Fingerprint (MD5):
  F6:24:EF:75:9D:B5:87:00:FC:19:2D:03:2C:18:C2:FE
 Fingerprint (SHA1):
  E1:47:A9:15:E2:89:9C:B1:F4:F3:02:C6:9A:7A:F3:CD:9C:4D:F7:95
```

Listing 5: Japanese certificate for electronic passports (with tool pp from nss)

10.4 NESSIE and ECRYPT

The European Union research project NESSIE (New European Schemes for Signatures, Integrity and Encryption) compiled a list of recommended cryptographic algorithms in 2004 [NESSIE, 2004] . It lists RSA-PSS exclusively and no variant with old padding. It should be noted however that the NESSIE recommendations are outdated and also contain at least one broken algorithm (SFLASH).

There is a follow-up research project to NESSIE called ECRYPT, which releases a "Yearly Report on Algorithms and Key Lengths". In the latest report, they list both RSA-PSS and RSA PKCS #1 v1.5, but write: "However, due to the lack of security proof, we recommend whenever possible to use RSA PSS instead, there is no advantage in using v1.5." (chapter 14.2.1, page 72 in [ECRYPT, 2010]) . They recommend not to use the same key for PSS and PKCS #1 v1.5.

10.5 CA/Browser Forum

Web browsers ship a list of root certificates from certificate authorities to validate the X.509 certificates from https connections. In 2007, the CA/Browser Forum defined guidelines for so-called Extended Validation (EV) certificates. These are certificates which have to comply with stricter security rules (it should be noted, however, that the EFF SSL Observatory found many EV certificates not compliant with their own rules).

The latest guidelines from the CA/Browser Forum contain no information about RSA padding rules [CA/Browser Forum, 2010] . They also claim that due to the lack of RSASSA-PSS implementations in browsers they have no plans to require PSS padding in the foreseeable future.

10.6 Summary

In many official state documents, at least a long term transition to PSS based RSA padding is scheduled. Internet certificate authorities don't seem to be upon the ones pushing for better cryptography.

The most ambitious plan comes from the German "Bundesnetzagentur", which require a full transition from hash-then-sign schemes to PSS until 2015 for messages and 2017 for certificates. However, it seems that there is no effort underway to accompany this time plan with a push for the required standards and implementations and there is a serious risk that if this doesn't happen very soon, it may be impossible to follow those requirements in practice.

11 Really provable Security

The concept of provable security we investigated with PSS and OAEP is a very limited one. It is only possible to provide "provable" security under certain assumptions. It relies on three assumptions: That factoring is a hard problem, that RSA is really as hard as factoring and that well-designed hash functions behave like random oracles. One could ask if it is at least theoretically possible to create provable secure public key cryptography only under provable assumptions.

What would that mean? We could define a "provable secure" public key function as one that can only be broken by an attacker that is able to do a certain number of calculations dependent on the key size. Now we can choose a key size high enough to make it implausible that any attacker with human technology may be able to break that it within, say, the lifetime of a human being.

What we would require is a proof that for any attacker skilled with the best possible algorithms, it requires a minimum amount of calculations to forge a signature for a given message and a public key or to get the decrypted message given an encrypted one and a public key. We will need some basic complexity theory before we can answer such questions.

11.1 Complexity Theory, P/NP and FP/FNP

Complexity theory defines so-called complexity-classes as sets of problems with certain properties.

The first complexity class we will investigate is P. A problem X is in P if it is a decision problem (the solution is just "yes" or "no") and an algorithm exists that solves X for any input with length n with a running time that can be expressed as a polynomial of n. Such algorithms are usually considered to be "fast". An example for a P problem are primality tests: "Given a number with n digits, can you decide if it is a prime?" For a long time, it was unknown if a polynomial primality test exists. In 2002, a polynomial primality algorithm – the AKS algorithm – was presented and thus showed that primality tests are in P.

The second important complexity class is NP. NP stands for "Nondeterministic Polynomial time", but we will use an easier notion here: NP problems are all decision problems where a polynomial algorithm exists to verify a result given some extra information.

The problems interesting for public key cryptography – like factoring – are usually not decision problems. But it is easily possible to transfer them into decision problems. A decision problem for factoring could be like this: "Given a large number N and a number X with $X < N$, is there a factor of N smaller than X?" A fast algorithm that is able to solve this decision problem can be used with a binary search (which itself is polynomial) to factor large numbers.

Problems that are not decision problems, but that can be transferred into an NP decision problem with a polynomial algorithm are called FNP. Analogous to that, problems that are not decision problems, but can be transferred into such by a polynomial algorithm are called FP.

It is easy to see that factoring is in FNP: If we have the factors of a large number N, we can easily check that by multiplying them. It is also easy to see that ANY problem that can be used for public key cryptography must be in FNP. The extra information here is the private key. Given someone has a private key, he can easily create forged signatures and decrypt messages – otherwise the algorithm would not make sense.

It is one of the big unsolved mathematical problems if $P \neq NP$ (the problem if $FP \neq FNP$ is equivalent). While it seems rather unlikely that $P = NP$, nobody has been able to prove that there actually exist problems that are in NP, but are harder than P. This is one of the seven millennium problems for which the Clay Mathematics Institute has designated one million dollar for the solution [Cook, 2000] .

Now as we have seen, all problems usable for public key cryptography are in FNP. If $P = NP$ then public key cryptography would not work at all. So a precondition to provable public key cryptography is the proof for $P \neq NP$.

11.2 NP complete Problems

Another relevant complexity class of problems are NP complete problems. If one had an efficient algorithm for any NP complete problem, this could be used to solve any other NP problem. So NP complete problems are the hardest problems inside the class of NP problems.

One might ask if it is possible to design a public key cryptosystem based on an NP-complete problem (or, to be more exact, on an FNP-complete problem). This has been tried various times in the past, but nobody has ever succeeded. Even if someone finds a suitable trapdoor function for an NP complete problem, this does not automatically mean that the system is secure. The reason is that complexity classes only make statements about the worst running time of an algorithm. For many NP complete problems, probabilistic algorithms exist that will not give optimal results in any case, but in the majority of cases. This makes it unusable for cryptosystems – an algorithm that provides security in a small number of cases but fails most of the time is of little use. To be able to use an NP-complete problem, we would need a way to ensure that the cryptosystem only uses the instances of an NP-complete problem that are actually hard to solve.

We have already mentioned two cryptosystems based on NP-complete problems in chapter 2.10. Most attempts to design an NP-complete cryptosystem have failed in the past - for example the Merkle-Hellmann knapsack cryptosystem [Shamir, 1984] (knapsack is an NP complete problem). A non-broken cryptosystem based on an NP-complete problem is the McEliece algorithm – it

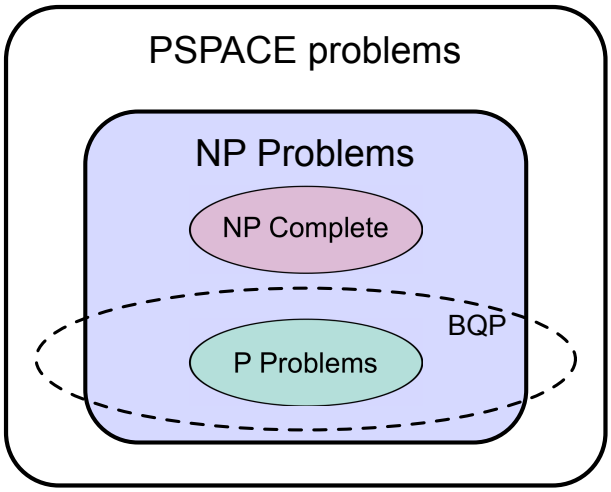


Figure 11: Expected (but unproven) relationship of complexity classes [16]

is rarely used because it has very large keys (for a sane parameter set around half a megabyte). This makes it unsuitable for most common applications, where it is a common task to transfer keys fast (for example, it is not unlikely that a simple https connection causes two or more RSA public keys to be transferred before any data is transmitted). But still, although McEliece is based on an NP complete problem, it lacks proof that it is as hard to break as the worst case scenarios of an NP complete problem - and that is what provable security would require.

11.3 Quantum Computers and BQP

A topic that has gained interest in recent years are quantum computers. Peter Shor [Shor, 1996] was able to show that with a quantum computer, it would be possible to solve certain problems relevant for all of today's mainstream public key systems in polynomial time. The Shor algorithm is able to exponentially reduce the complexity and running time of factoring and discrete logarithm algorithms. Shor's algorithm can also be applied to discrete logarithms in elliptic curves. So this means that if anyone was able to build a large enough quantum computer, it would allow breaking RSA, Rabin-Williams, ElGamal, DSA and ECDSA. However, there is no need to panic yet: The best quantum computer built up until now had 7 quantum bits and was successfully able to factor the number 15 into 3 and 5 [IBM, 2001] . The class of those problems is called BQP (bounded error quantum polynomial time).

A quantum computer is using physical effects of quantum theory to solve problems that presumably cannot be solved efficiently on classical computers. So this brings in an important topic: If we want to construct a provable secure

[16] Diagram from Wikimedia Commons at `http://commons.wikimedia.org/wiki/File:BQP_complexity_class_diagram.svg`

public key cryptosystem, we want it not only to be unbreakable by classical computers, but preferably unbreakable by all computers that can be built within our physical universe. This makes the question of provable security not only a problem of complexity theory and mathematics, but also a problem of physics.

A paper by Scott Aaronson discusses the question of complexity and physical reality in detail [Aaronson, 2005] . He introduces the "NP Hardness Assumption" – the assumption that NP complete problems are intractable in the physical world.

11.4 Provable secure Public Key Algorithm

Concluding the above, to construct a provable secure public key cryptosystem, we would need to do the following steps:

- Show that P != NP.
- Define and prove a lower bound of complexity for a given NP problem (we will call it LP). This may be an NP complete problem or not, depending very much on what the proof for P != NP and for lower complexity bounds looks like.
- Find a trapdoor function for LP.
- Construct a public key cryptosystem out of LP.
- Prove that anyone who can break KLP can also solve LP, which would imply KLP is as secure as it is hard to solve LP.
- Prove that KLP does always map to the hard parts of LP and that no probabilistic efficient algorithms exist.
- Show that solving LP can not be accelerated significantly by the use of physical effects like quantum computers.

Finally, if we want to construct real-world cryptography out of KLP, we would probably also need hash functions and symmetric ciphers with provable properties, as cryptosystems are usually hybrid constructions. Also, we need schemes like PSS that provide provable security for the whole constructions.

11.5 Summary

Extending the idea of provable security to a concept that is not limited to model assumptions like the random oracle model or that factorization is hard seems to be far out of reach with today's knowledge in complexity theory.

The first step ($P! = NP$) is considered being one of the hardest theoretical problems in computer science and mathematics at all and no solution to it is in sight. Several of the other steps require breakthroughs in areas of mathematics

and physics where today very little is known at all. So while what would be required for "provable security" is in theory imaginable, it is unlikely to happen any time soon.

12 Conclusion

12.1 Difficulties in deploying better Cryptography

In theory, we'd like to see the best cryptography that is available used within real world applications. In practice, this is often not the case – often enough, what's used is just "what's not completely broken yet" and widespread adoption waits until real-world attacks can be shown in practice.

We saw that the PSS padding provides better security in certain circumstances than old padding methods like PKCS #1 v1.5. However, adoption of RSA-PSS is widely lacking. What we can see here and also with similar examples (like the transition from MD5/SHA-1 to SHA-2) is that it seems to be very hard to deploy improvements from theoretical cryptographic research to practical applications.

For an algorithm to be used, it usually has to pass several obstacles:

1. The algorithm has to be developed, examined and accepted by the cryptographic community.
2. The algorithm primitives have to be standardized.
3. High level protocols using those primitives need to be standardized.
4. The standards need to be implemented in libraries and applications, both the primitives and the high level protocols.
5. Finally, they need to be used and set as the default within applications.

RSASSA-PSS is mostly stuck between 2 and 3. But even when looking at the example of SHA-2, it is in many cases stuck between 3 and 4, although real-world attacks are expected within the next couple of years.

One of the most often cited reasons not to switch to better algorithms is backwards compatibility. As an extreme example, within the browser marked, it usually can be expected that rarely anyone will adopt anything that is not backwards compatible to browsers several years old (the argument I hear most often when asking for signatures using SHA-2 instead of SHA-1 is that Windows XP before SP3 does not support that). So even if at some point all browsers support X.509 certificates with RSASSA-PSS, it is unlikely that any major certificate authority will use them any time soon.

But that does not explain why it also seems to be very hard to deploy improvements on the standards level. As we have seen, PSS has not been

adopted in a couple of new standards like DNSSEC or DKIM that were developed after the RSASSA-PSS primitives had been standardized in PKCS #1 v2.1. An important argument here was that the standardization committees feared their standards would not be accepted at all if they went with a cryptographic primitive that hash limited support in current software. A typical chicken-egg-problem between software and standards.

The biggest push for new algorithms may come from public authorities. In the case of PSS, if the German BSI insists on its plans to require it after 2015 for qualified signatures, it is likely that this will push implementations within all major email clients. Similar rules could be applied on other applications, for example https connections and their certificates on crucial government web services.

Finally, the people working on those issues are very different and often have a limited understanding of the areas of the others. Internet protocol designers and software developers often do not have any cryptographic expertise. On the other hand cryptographers are often mathematicians or theoretical scientists who do not have much experience with programming and even less with operating real-world internet services. More interdisciplinary collaboration would be desirable.

Looking at security threats causing real-world attacks, the vast majority does not involve cryptography at all (buffer overflows, SQL injections). From those that involve cryptography, implementation problems or high-level protocol design flaws are much more likely to be exploited than problems in the base algorithm primitives. So for software developers, improving security on that layer comes with a very low priority.

12.2 Summary

We have presented the RSA-PSS padding scheme. Beside the better theoretical properties, we have sown also that due to the randomization involved, RSA-PSS provides more robustness with respect to real-world problems that happened in the past. Despite that, we saw that 15 years after its invention and eight years after having been standardized, we don't see any relevant use of it. This may change in some areas as requirements by public authorities within some years will require the adoption of PSS.

In all new applications, whenever possible RSASSA-PSS should be preferred over EMSA-PKCS1-v1_5. For already existing applications, a gradual transition is advisable. As there is no direct threat, there is no need to rush on that, but it is a useful extra security consideration. Secure choices for key size (minimum 2048 bit), exponent (65537 or above) and hash function (SHA-2 family or upcoming SHA-3 standard) should be self-evident. To achieve an even better security, a combination of RSASSA-PSS with randomized hashing should be considered.

To conclude things, I doubt that PSS will see any widespread usage at all. When considering the switch to new public key algorithms, most people today

are more likely to consider the use of elliptic curve cryptography. When considering the future of public key cryptography in the long term, we will probably see another topic raising within the next years: The requirement to have algorithms resistant to quantum computers. Neither PSS nor elliptic curves (nor anything else deployed today) provides anything here, so in the long run we will probably see completely new public key algorithms developed.

What stays is the general idea of provable security. While it is far from being realistic to see any system that is fully provably secure, it seems to be a reasonable approach to provide provable security in parts of systems where it is possible. When we cannot prove the security of a full system, we can at least try to prove parts of the system and show if the complexity of systems relies on well-known and understood problems. Also, we saw that the PSS construction provides more robustness against flaws in the implementation. It seems reasonable to follow that approach further and investigate algorithm designs that are less likely to be implemented with security flaws.

References

[Aaronson, 2005] Aaronson, S. (2005). NP-complete Problems and Physical Reality. Available from: http://www.scottaaronson.com/papers/npcomplete.pdf.

[Aumasson et al., 2010] Aumasson, J.-P., Henzen, L., Meier, W., and Phan, R. C.-W. (2010). SHA-3 proposal BLAKE. Available from: http://131002.net/blake/blake.pdf.

[Bellare and Rogaway, 1993] Bellare, M. and Rogaway, P. (1993). Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. Available from: http://cseweb.ucsd.edu/users/mihir/papers/ro.pdf.

[Bellare and Rogaway, 1995] Bellare, M. and Rogaway, P. (1995). Optimal Asymmetric Encryption – How to Encrypt with RSA. Available from: http://www-cse.ucsd.edu/users/mihir/papers/oae.pdf.

[Bellare and Rogaway, 1996] Bellare, M. and Rogaway, P. (1996). The exact security of digital signatures: How to sign with RSA and Rabin. Available from: http://www.cs.ucdavis.edu/~rogaway/papers/exact.html.

[Bellare and Rogaway, 1998] Bellare, M. and Rogaway, P. (1998). PSS: Provably secure encoding method for digital signatures. Available from: http://www.cs.ucdavis.edu/~rogaway/papers/exact.html.

[Bellare and Rogaway, 2001] Bellare, M. and Rogaway, P. (2001). United States Patent 6,266,771 – Probabilistic Signature Scheme. Available from: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=6,266,771.PN.&OS=PN/6,266,771&RS=PN/6,266,771.

[Bellare and Rogaway, 2006] Bellare, M. and Rogaway, P. (2006). United States Patent 7,036,014 – Probabilistic Signature Scheme. Available from: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=7,036,014.PN.&OS=PN/7,036,014&RS=PN/7,036,014.

[Bernstein, 2009] Bernstein, D. J. (2009). Introduction to post-quantum cryptography. Available from: http://pqcrypto.org/www.springer.com/cda/content/document/cda_downloaddocument/9783540887010-c1.pdf.

[Boneh et al., 1996] Boneh, D., DeMillo, R. A., and Lipton, R. J. (1996). On the importance of checking cryptographic protocols for faults. Available from: http://crypto.stanford.edu/~dabo/abstracts/faults.html.

[Boneh and Shao, 2007] Boneh, D. and Shao, W. (2007). Randomized Hashing for Digital Certificates: Halevi-Krawczyk Hash. Available from: http://crypto.stanford.edu/firefox-rhash/.

[Brier et al., 2006] Brier, E., Chevallier-Mames, B., Ciet, M., and Clavier, C. (2006). Why one should also secure RSA public key elements. Available from: http://www-mlab.jks.ynu.ac.jp/ches/Eric%20Brier.pdf.

[Brown, 2005] Brown, D. R. L. (2005). A Weak-Randomizer Attack on RSA-OAEP with e = 3. Available from: http://eprint.iacr.org/2005/189.

[BSI, 2011] BSI (2011). Technische Richtlinie 03125 – Beweiswerterhaltung kryptographisch signierter Dokumente – Anlage TR-ESOR-M.2: Krypto-Modul – Version 1.1. Available from: https://www.bsi.bund.de/ContentBSI/Publikationen/TechnischeRichtlinien/tr03125/index_htm.html.

[Bundesnetzagentur, 2011] Bundesnetzagentur (2011). Algorithmenkatalog: Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen). Available from: http://www.bundesnetzagentur.de/cae/servlet/contentblob/192414/publicationFile/10008/2011AlgoKatpdf.pdf.

[CA/Browser Forum, 2010] CA/Browser Forum (2010). Guidelines for the Issuance and Management of Extended Validation Certificates 1.3. Available from: http://www.cabforum.org/Guidelines_v1_3.pdf.

[Canetti et al., 2002] Canetti, R., Goldreich, O., and Halevi, S. (2002). The Random Oracle Methodology, Revisited. Available from: http://eprint.iacr.org/1998/011.pdf.

[Cook, 2000] Cook, S. (2000). The P versus NP Problem. Available from: http://www.claymath.org/millennium/P_vs_NP/pvsnp.pdf.

[Coron et al., 2009] Coron, J.-S., Joux, A., Naccache, D., and Paillier, P. (2009). Fault Attacks on Randomized RSA Signatures. Available from: http://www.jscoron.fr/publications/iso2fault.pdf.

[Coron and Mandal, 2009] Coron, J.-S. and Mandal, A. (2009). PSS is secure against random fault attacks. Available from: http://www.jscoron.fr/publications/pssfault.pdf.

[Dang, 2009] Dang, Q. (2009). NIST Special Publication 800-106 – Randomized Hashing for Digital Signatures. Available from: http://csrc.nist.gov/publications/nistpubs/800-106/NIST-SP-800-106.pdf.

[Davida, 1982] Davida, G. (1982). Chosen signature cryptanalysis of the RSA (MIT) public key cryptosystem.

[Dent, 2006] Dent, A. W. (2006). Fundamental problems in provable security and cryptography. Available from: http://eprint.iacr.org/2006/278.pdf.

[Diffie and Hellman, 1977] Diffie, W. and Hellman, M. E. (1977). New Directions in Cryptography. Available from: http://groups.csail.mit.edu/cis/crypto/classes/6.857/papers/diffie-hellman.pdf.

[Dobbertin, 1996] Dobbertin, H. (1996). Cryptoanalysis of MD5 Compress. Available from: http://www.iacr.org/conferences/ec96/rump/dobberti.ps.gz.

[ECRYPT, 2010] ECRYPT (2010). European Network of Excellence in Cryptology II: Yearly Report on Algorithms and Keysizes (2009-2010). Available from: http://www.ecrypt.eu.org/documents/D.SPA.13.pdf.

[ElGamal, 1985] ElGamal, T. (1985). A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. Available from: http://groups.csail.mit.edu/cis/crypto/classes/6.857/papers/elgamal.pdf.

[ETSI, 2007] ETSI (2007). ETSI TS 102 176-1 V2.0.0 – Electronic Signatures and Infrastructures (ESI); Algorithms and Parameters for Secure Electronic Signatures; Part 1: Hash functions and asymmetric algorithms. Available from: http://www.etsi.org/deliver/etsi_ts/102100_102199/10217601/02.00.00_60/ts_10217601v020000p.pdf.

[EU, 1999] EU (1999). DIRECTIVE 1999/93/EC – Community framework for electronic signatures. Available from: http://eur-lex.europa.eu/smartapi/cgi/sga_doc?smartapi!celexapi!prod!CELEXnumdoc&numdoc=31999L0093&model=guichett.

[Finney, 2006] Finney, H. (2006). Bleichenbacher's RSA signature forgery based on implementation error. Available from: http://www.mail-archive.com/cryptography@metzdowd.com/msg06537.html.

[Gauravaram et al., 2011] Gauravaram, P., Knudsen, L. R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., and Thomsen, S. S. (2011). Grøstl – a SHA-3 candidate. Available from: http://www.groestl.info/Groestl.pdf.

[Grell and University of California, 1999] Grell, M. and University of California (1999). Mail to IEEE P1363 committee about PSS patents. Available from: http://grouper.ieee.org/groups/1363/P1363/letters/UC.html.

[Halevi and Krawczyk, 2007] Halevi, S. and Krawczyk, H. (2007). Strengthening Digital Signatures via Randomized Hashing. Available from: http://webee.technion.ac.il/~hugo/rhash/rhash.pdf.

[IBM, 2001] IBM (2001). IBM's Test-Tube Quantum Computer Makes History. Available from: http://www-03.ibm.com/press/us/en/pressrelease/965.wss.

[ICAO, 2006] ICAO (2006). Machine Readable Travel Documents – Part 1: Machine Readable Passports – Volume 2: Specifications for Electronically Enabled Passports with Biometric Identification Capability. Available from: http://www2.icao.int/en/MRTD/Downloads/Doc%209303/Doc%209303%20English/Doc%209303%20Part%203%20Vol%202.pdf.

[IEEE, 2004] IEEE (2004). P1363a – Standard Specifications for Public Key Cryptography — Amendment 1: Additional Techniques. Available from: http://grouper.ieee.org/groups/1363/P1363a/.

[IETF Network Working Group, 2005a] IETF Network Working Group (2005a). RFC 4055 – Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Available from: http://tools.ietf.org/html/rfc4055.

[IETF Network Working Group, 2005b] IETF Network Working Group (2005b). RFC 4056 – Use of the RSASSA-PSS Signature Algorithm in Cryptographic Message Syntax (CMS). Available from: http://tools.ietf.org/html/rfc4056.

[IETF Network Working Group, 2006] IETF Network Working Group (2006). RFC 4359 – The Use of RSA/SHA-1 Signatures within Encapsulating Security Payload (ESP) and Authentication Header (AH). Available from: http://tools.ietf.org/html/rfc4359.

[IETF Network Working Group, 2007] IETF Network Working Group (2007). RFC 4880 – OpenPGP Message Format. Available from: http://tools.ietf.org/html/rfc4880.

[IETF Network Working Group, 2009] IETF Network Working Group (2009). RFC 5702 – Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC. Available from: http://tools.ietf.org/html/rfc5702.

[IETF Network Working Group, 2010] IETF Network Working Group (2010). RFC 5756 – Updates for RSAES-OAEP and RSASSA-PSS Algorithm Parameters. Available from: http://tools.ietf.org/html/rfc5756.

[ISO, 2006] ISO (2006). ISO 18033-2: A Standard for Public-Key Encryption. Available from: http://www.shoup.net/iso/.

[Kleinjung et al., 2010] Kleinjung, T., Aoki, K., Franke, J., Lenstra, A. K., Thomé, E., Bos, J. W., Gaudry, P., Kruppa, A., Montgomery, P. L., Osvik, D. A., te Riele, H., Timofeev, A., and Zimmermann, P. (2010). Factorization of a 768-bit RSA modulus. Available from: http://eprint.iacr.org/2010/006.pdf.

[Lanz et al., 2007] Lanz, K., Bratko, D., and Lipp, P. (2007). RSA-PSS in XMLDSig. Available from: http://www.w3.org/2007/xmlsec/ws/papers/08-lanz-iaik/.

[McEliece, 1978] McEliece, R. J. (1978). A Public-Key Cryptosystem based on algebraic Coding Theory. Available from: http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[Microsoft, 2010] Microsoft (2010). Validating the Certificate Chain. Available from: http://msdn.microsoft.com/en-us/library/dd407310(v=vs.85).aspx.

[NESSIE, 2004] NESSIE (2004). New European Schemes for Signatures, Integrity and Encryption: Final report of European project number IST-1999-12324. Available from: https://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf.

[NIST, 2010] NIST (2010). NIST Special Publication SP 800-78-3 – Cryptographic Algorithms and Key Sizes for Personal Identity Verification. Available from: http://csrc.nist.gov/publications/nistpubs/800-78-3/sp800-78-3.pdf.

[Rabin, 1979] Rabin, M. O. (1979). Digitalized signatures and public-key functions as intractable as factorization. Available from: http://publications.csail.mit.edu/lcs/specpub.php?id=780.

[Rivest et al., 1977] Rivest, R. L., Shamir, A., and Adleman, L. (1977). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Available from: http://people.csail.mit.edu/rivest/Rsapaper.pdf.

[RSA Inc., 1993] RSA Inc. (1993). PKCS #1 v1.5. Available from: http://www.rsa.com/rsalabs/node.asp?id=2125.

[RSA Inc., 2002] RSA Inc. (2002). PKCS #1 v2.1. Available from: http://www.rsa.com/rsalabs/node.asp?id=2125.

[RSA Inc., 2004] RSA Inc. (2004). PKCS #11 v2.20. Available from: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf.

[Schneier, 1999] Schneier, B. (1999). Elliptic Curve Public-Key Cryptography. Available from: http://www.schneier.com/crypto-gram-9911.html#EllipticCurvePublic-KeyCryptography.

[Schneier et al., 2010] Schneier, B., Ferguson, N., Lucks, S., Whiting, D., Bellare, M., Kohno, T., Callas, J., and Walker, J. (2010). The Skein Hash Function Family – Version 1.3. Available from: http://www.skein-hash.info/sites/default/files/skein1.3.pdf.

[Schneier et al., 1997] Schneier, B., Kelsey, J., and Wagner, D. (1997). Protocol Interactions and the Chosen Protocol Attack. Available from: http://www.schneier.com/paper-chosen-protocol.html.

[Shamir, 1984] Shamir, A. (1984). A polynomial-time algorithm for breaking the basic Merkle - Hellman cryptosystem.

[Shamir and Tromer, 2003] Shamir, A. and Tromer, E. (2003). Factoring Large Numbers with the TWIRL Device. Available from: http://people.csail.mit.edu/%7Etromer/papers/twirl.pdf.

[Shor, 1996] Shor, P. (1996). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. Available from: http://arxiv.org/abs/quant-ph/9508027v2.

[Sotirov et al., 1998] Sotirov, A., Stevens, M., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D. A., and de Weger, B. (1998). MD5 considered harmful today. Available from: http://media.ccc.de/browse/congress/2008/25c3-3023-en-making_the_theoretical_possible.html.

[Technical Committee CEN/TC 224, 2008] Technical Committee CEN/TC 224 (2008). prEN 14890-1:2008 – Application Interface for smart cards used as Secure Signature Creation Devices – Version 2.2.

[ticalc.org, 2009] ticalc.org (2009). TI-83 Plus OS Signing Key Cracked. Available from: http://www.ticalc.org/archives/news/articles/14/145/145154.html.

[US-CERT, 2008] US-CERT (2008). Vulnerability Note VU#800113 – Multiple DNS implementations vulnerable to cache poisoning. Available from: http://www.kb.cert.org/vuls/id/800113.

[W3C, 2002] W3C (2002). XML Encryption Syntax and Processing. Available from: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.

[W3C, 2008] W3C (2008). XML Signature Syntax and Processing (Second Edition). Available from: http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/.

[Wang et al., 2004] Wang, X., Feng, D., Lai, X., and Yu, H. (2004). Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Available from: http://eprint.iacr.org/2004/199.

[Wang et al., 2005] Wang, X., Yin, Y. L., and Yu, H. (2005). Finding Collisions in the Full SHA-1. Available from: http://people.csail.mit.edu/yiqun/SHA1AttackProceedingVersion.pdf.

[Williams, 1980] Williams, H. (1980). A modification of the RSA public-key encryption procedure.

Nomenclature

API Application Programming Interface

ASN.1 Abstract Syntax Notation One

BQP Bounded error Quantum Polynomial time

BSI Bundesamt für Sicherheit in der Informationstechnik

CSCA Country Signing Certificate Authority

DKIM DomainKeys Identified Mail

DNSSEC Domain Name System Security Extension

DSA Digital Signature Algorithm

eTCR enhanced Target Collision Resistance

ETSI European Telecommunications Standards Institute

EV Extended Validation

FDH Full Domain Hashing

GPG GNU Privacy Guard

GPL GNU General Public License

ICAO International Civil Aviation Organization

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IPsec Internet Protocol Security

ISO International Organization for Standardization

KDF2 Key Derivation Function 2

LGPL GNU Lesser General Public License

MD5 Message Digest 5

MGF1 Mask Generation Function 1

MPL Mozilla Public License

NESSIE New European Schemes for Signatures, Integrity and Encryption

NIST National Institute of Standards and Technology

NP Nondeterministic Polynomial Time

NSA National Security Agency

nss Network Security Services

OAEP	Optimal asymmetric Encryption Padding
OID	Object Identifier
P	Polynomial Time
PGP	Pretty Good Privacy
PKCS	Public Key Cryptography Standards
PSS	Probabilistic Signature Scheme
RFC	Request for Comments
RSA	Rivest, Shamir, Adelman Algorithm
RW	Rabin-Williams
SHA	Secure Hash Algorithm
SNI	Server Name Indication
SSA	Signature Scheme with Appendix
SSL	Secure Socket Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den

Johannes Böck

Einverständniserklärung

Ich erkläre hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Institutes für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den

Johannes Böck