A Discrete Logarithm Hash Function for RSA Signatures

Ralf Senderek

Abstract

To reduce the complexity of a cryptosystem it can be useful to have the hash function's security based on the same foundation as the public key encryption scheme. In this respect the proposal of a discrete logarithm hash function once invented by Adi Shamir offers both a clear concept and provable collision-resistance. With both cryptographic primitives based on modular exponentiation the possibility of dangerous interaction between the hash and the signature scheme can be avoided reliably once the user's key material is carefully selected. This paper focuses on the security implications which have to be considered when the discrete logarithm hash is used to create RSA signatures, a purpose, for which it proves to be particularly useful.

1 Introduction

Real-world cryptosystems have become very complex. Not only numerous kinds of complex features have been built into the cryptosystems like subkeys, message recovery keys, automated key server requests etc., which on the one hand have made cryptosystems more comfortable and sometimes more insecure [S-2000], but on the other hand a valuable characteristic has been sacrificed to the impressive progress of modern cryptography, the ordinary user's understanding of what is really going on. Usually the basic components of a cryptosystem at least comprise a public key mechanism based on the infeasibility of factoring or computing discrete logarithms, a symmetric cipher to speed up the encryption process and a standard hash algorithm which is used to create signatures.

A solution to the complexity problem might make it neccessary to reduce the number of different crypto-algorithms to a bare minimum, giving preference to the most concise and clear concepts available in the field. If one does not worry about performance, there is really no need for a symmetric cipher, which will burden the cryptosystem with a number of avoidable security weaknesses or risks only. Given a proper padding of the plaintext [Bon] and sufficient key lengths, a public key cipher like RSA can do all encryption with a simple, intuitive operation, modular exponentiation. This paper will explore the possibility to perform the hashing with a similar intuitive mechanism, first proposed by Adi Shamir, which is not only a clear concept but comes with a prove of collision-resistance, contemporary standard hash functions all are missing.

2 Adi Shamir's proposal and the prove of collision-resistance

The hash function originally proposed by Adi Shamir years ago, relies on one single step of modular exponentiation. Once the message has been converted into a long integer, a hash of the message can be computed by

$$hash(x) = g^x \; mod \, n \tag{1}$$

given that both p and q are large primes which are being kept secret so that factoring of the modulus $n = p * q$ is computationally infeasible.

This hash function is provably collision-resistant, as Ronald L. Rivest pointed out in a posting which I quote here [R-2003].

```
"Adi Shamir once proposed the following hash function:

     Let n = p*q be the product of two large primes, such that
     factoring n is believed to be infeasible.

     Let g be an element of maximum order in Z_n^* (i.e. an
     element of order lambda(n) = lcm(p-1,q-1)).

     Assume that n and g are fixed and public; p and q are secret.

     Let x be an input to be hashed, interpreted as a
     non-negative integer.  (Of arbitrary length; this may be
     considerably larger than n.)

     Define hash(x) = g^x (mod n).

Then this hash function is provably collision-resistant, since
the ability to find a collision means that you have an x and
an x' such that

     hash(x) = hash(x')

which implies that

     x - x' = k * lambda(n)

for some k.  That is a collision implies that you can find a
multiple of lambda(n).  Being able to find a multiple of lambda(n)
means that you can factor n."
```

To complete the picture one has to bear in mind that Euler's theorem states that for every g which is relatively prime to $\phi(n) = (p-1) * (q-1)$ the following equation holds.

$$g^{\phi(n)} \; mod \; n = 1 \tag{2}$$

As $\phi(n)$ is a multiple of $\lambda(n)$, we have $k * \lambda(n) = k' * \phi(n) = k' * t * \lambda(n)$ for some integers k′ and t.

For all k′ that divide k we can compute a collison using Euler's theorem, because

$$x = x' + k' * \phi(n)$$

and

$$\begin{aligned} hash(x) &= g^{(x' + k' * \phi(n))} \; mod \; n \\ &= g^{x'} * g^{k' * \phi(n)} \; mod \; n \\ &= g^{x'} * \left(g^{\phi(n)}\right)^{k'} \; mod \; n \end{aligned}$$

with (2) follows

$$\begin{aligned} hash(x) &= g^{x'} * 1^{k'} \; mod \; n \\ &= g^{x'} \; mod \; n \\ hash(x) &= hash(x') \end{aligned}$$

In their original description the authors of the RSA cryptosystem argued that computing $\phi(n)$ and factoring n are equally hard problems [RSA-78, section IX B] since $\phi(n)$ can be used to factor n. With factoring of n being computationally infeasible, only an exhaustive search remains to find $\phi(n)$.

3 The complexity of a brute force attack on $\phi(n)$

Of course an attacker wanting to find a collision of the hash function for a given input message x will be successful once he has found $\lambda(n)$, which in general is a much easier problem to solve than finding $\phi(n)$, but as we can use "Strong Primes" [RS-98, see section 4] to build the hash function's modulus, the range in which an exhaustive search for a multiple of $\lambda(n)$ is successful will not be considerably smaller than the range in which $\phi(n)$ will be found.

A "strong prime" (p^--strong) is a prime which is a large prime , i.e. its binary length is $|p| \geq 500$ and the largest prime factor of $p - 1$ is also large, so that both $\phi(n)$ and $\lambda(n)$ will be at least 1000 bit long.

Let p and q be two p^--strong primes of comparable length, i.e

$$q = p + d \qquad\qquad (0 \leq d \leq p)$$

We estimate the upper bound of the range R in which $\phi(n)$ will be found assuming that p and q are almost of the same length, so that the difference d is close to 0. From $|p| = |q| = \sqrt{n}$ follows

$$\begin{aligned} \phi(n)_{max} &= (\sqrt{n} - 1) * (\sqrt{n} - 1) \\ &= n - 2\sqrt{n} + 1 \end{aligned}$$

While the difference between p and q increases, $\phi(n)$ decreases until it reaches its lower bound when $q = 2\,p$ and $p = \sqrt{\frac{n}{2}}$ at

$$\begin{aligned} \phi(n)_{min} &= (\sqrt{\frac{n}{2}} - 1) * (2\sqrt{\frac{n}{2}} - 1) \\ &= n - 3\sqrt{\frac{n}{2}} + 1 \end{aligned}$$

Therefore the difference between $\phi(n)_{max}$ and $\phi(n)_{min}$ will be

$$\begin{aligned} r &= \phi(n)_{max} - \phi(n)_{min} \\ &= 3\sqrt{\frac{n}{2}} - 2\sqrt{n} \\ &= \sqrt{\frac{n}{2}} * \left(3 - 2\sqrt{2}\right) \\ &= \sqrt{n} * \left(\frac{3}{\sqrt{2}} - 2\right) \\ r &> 1/8 * \sqrt{n} \end{aligned}$$

Thus an exhaustive search for $\phi(n)$ in the range $R = [\phi_{min}, ..., \phi_{max}]$ will require more than 2^{497} steps and is clearly beyond todays available computational abilities even if the requirements for the length of the prime factor in $(p - 1)$ and $(q - 1)$ are reduced.

4 Interactions between the hash function and the RSA signature scheme

So far we have been focusing on the two major advantages which follow from the design of the discrete logartithm hash function, the clear concept of exponentiation and its collision-resistance. On the other hand we can expect that the dependance on the same fundamental operation (modular exponentiation) will cause negative implications once the hash is used to create signatures with the RSA signature scheme [Dam-88, p. 214]. It is important that these obvious concerns can be destroyed when certain precautions are taken.

With RSA cryptosystems a signature of a message is created with a secret signing key $K^{-1} = [d, N = p' * q']$ and a public key $K = [e, N]$ is used to verify the signature. Usually a hash value of the message text is signed to guard against the multiplicative homomorphism of the RSA signature scheme. Without this protection it would be possible for everyone to create new valid signatures simply by multiplication of any number of old signatures (mod N). This requires the hash function to be multiplication free [And-93, section 3.3].

A signature is created using the secret decryption exponent d by computing

$$sig(x) = [hash(x)]^d \; mod\, N. \tag{3}$$

To verify this signature with the public encryption exponent e one computes

$$\begin{aligned} [sig(x)]^e \; mod\, N &= [hash(x)]^{e*d} \; mod\, N \\ &= hash(x) \end{aligned} \tag{4}$$

which recovers a hash value that is easily compared with a fresh hash of the message in question.

Although the hash function is not multiplication free by design we can show that it is infeasible to exploit this characteristic to create forged signatures once the modulus of the signature scheme (N) is different from the modulus (n) used for hashing.

Let x and y be some arbitrary inputs. Signatures on these values can then be computed

$$\begin{aligned} sig(x) &= (g^x \; mod\, n)^d \; mod\, N \\ &= (g^x - a_1 n)^d \; mod\, N \\ sig(y) &= (g^y \; mod\, n)^d \; mod\, N \\ &= (g^y - a_2 n)^d \; mod\, N \end{aligned}$$

with a_1 and a_2 being two very large integer values. A different signature $sig(z)$ can subsequently be obtained through multiplication of old signatures.

$$\begin{aligned} sig(z) &= sig(x) * sig(y) \; mod\, N \\ &= (g^x - a_1 n)^d * (g^y - a_2 n)^d \; mod\, N \\ &= [(g^x - a_1 n) * (g^y - a_2 n)]^d \; mod\, N \\ &= \left[g^{x+y} + n\,[a_1 a_2 n - (a_2 g^x + a_1 g^y)]\right]^d \; mod\, N \\ &= \left[g^{x+y} + n * t\right]^d \; mod\, N \\ sig(z) &= \left[\left(g^{x+y}\right)^d + n * k\right] \; mod\, N \end{aligned} \tag{5}$$

for some very large unknown integer t and k.

Bad interaction between the signature and hashing algorithms is clearly inevitable if both moduli are equal, i.e. if $p * q = p' * q'$, since

$$\begin{aligned} sig(z) &= \left[g^{x+y}\right]^d \; mod\, N \\ &= [hash(x+y) + a_3 n]^d \; mod\, N \\ &= [hash(x+y)]^d \; mod\, N \\ &= sig(x+y) \end{aligned}$$

Thus the moduli used for signing and hashing must be different.

In case both moduli are different, bad interaction can also occur, but as the value of the large unknown integer k depends on the chosen input values only, there are two different cases in which it is infeasible to select both inputs to match the conditions for bad interaction.

Case 1 : N and n have one common prime factor, say $p = p'$.

In this case bad interaction requires that $k * q$ is a multiple of q' which implies that q' divides k. To ensure this one must be able to factor N to gain q' which is supposed to be infeasible.

Case 2 : N and n have no common prime factors.

In this case bad interaction requires that $k*n$ is a multiple of N which implies that N divides k. This will work without knowing the factorisation of N, but it implies that the inputs are selected such that k has both large unknown p' and q' as prime factors. The polynomial, that determines k is of degree $d - 1$, so that solving (5) for a multiple of N is hard, if d is almost of equal length as is N.

5 The hash function's dependence on key material

5.1 Hash keys

With the use of a hash modulus a new concept is introduced into hashing, which seems to make the discrete logarithm hash more complicated compared to the standard hash functions, whose output depends on the message to be hashed only. In fact making the output depending on a user's hash key as well has several implications that must be considered.

The most obvious implication results from the fact that the person who produces the hash modulus clearly has an advantage over everyone who uses the hash function [Pre-94, p.19], because this person can factor the modulus and has the ability to create collisions. In my view this implies that it would be imprudent to rely on a trusted third party to create a single hash modulus (and generator) to avoid the dependence of the hash output on individual key material. On the contrary one should see the dependence on individual key material as an advantage of the hash, because the user himself can guarantee the security of the hash output as the selection process of his individual hash modulus and generator is entirely under his own control. This of course implies that somebody creating individual hash key material must be fully informed about all the requirements, and applications using the hash must check not only the integrity of the user's hash key, which would also include the binding to the user's identity, but also the length of the key material to reject moduli that can be factored according to recent knowledge about factoring. For everyone who verifies a signature the fact, that a person has created his own hash modulus is perfectly acceptable, as the person could easily sign a different message with his secret key. And for the person himself keping his primes secret is the guarantee that nobody else will be capable of constructing collisions, except by exhaustive search.

Another objection to use a hash modulus is that by chance two individuals may chose the same modulus although their generator values may be different. Cryptosystems using individual hash keys will require to check the key database for common moduli but that would not exclude the very small risc of common moduli in use when key databases are maintained locally on the user's system instead of a publicly available database like key servers. But this risc corresponds to the use of RSA keys as well, where common moduli lead to the ability to forge a signature directly.

Using an individual hash modulus bears the further advantage that it makes birthday attacks much more difficult, because finding a pair of different messages that hash to the same value would not only require a lot of precomputation but would also work for one particular hash key only so that the whole process has to be repeated for every other hash key.

5.2 Recommendations

When a user creates his hash key the following conditions should be met:

1) The primes p and q forming the hash modulus must be different from the primes p' and q' forming the public signing key's modulus.
   In general this condition is met when both moduli have considerably different size.
2) Both primes p and q should differ by 1 bit in size. If the difference between p and q is too small, the value $\phi(n)$ may be found close to the upper boundary described in section 3.
3) If possible p^--strong primes shall be chosen to form the hash modulus.
   A method to obtain such primes is described in [RS-98, section 5]. It basically starts with a randomly chosen integer (p^-) that tests positive for pimality, subsequently checking $p = a * p^- + 1$ until a prime is found.
4) All created primes must be kept secret.

And finally:

5) The generator value g should be of maximum order to increase the difficulty of calculating discrete logarithm values directly and to thwart precomputation attacks. It should be a generator of a prime order group to avoid weaknesses Ross Anderson has described for DSS and the Diffie-Hellman protocol emerging from the use of smooth subgroups due to the selection of the generator value g. See [AV, section 2.3 and 3.1] for details.

6 Reduction of the hash values

Standard hash functions usually produce an output value of fixed length, typically 160 bit or 256 bit long. As the length of the discrete logarithm hash values depends on the hash modulus alone it will vary, making it difficult for applications to integrate the hash value into their internal data structure. Thus the demand for a fixed length (i.e 256 bit) must be met for the discrete logarithm hash as well.

To ensure, that every bit of the long hash value contributes to the fixed-length output value I recommend to split the long output into sections of 256 bit linking all sections together with an XOR-operation. This is common practice in cryptosystems that derive a keystream from a number of pointers into random data [FSW, section 2] and should work for the reduction of the hash value size as well.
Although the reduction of the hash size may be required in special circumstances, it is not necessary to create signatures as the public signing key would usually be longer than the hash key, because as a long term key it would probably have an extra security margin to make factoring as hard as possible.

7 Conclusions

We have seen that using the discrete logarithm hash function together with the RSA signature scheme is not only possible but will produce signatures whose security is based on the discrete log problem and on the infeasibility to factor large numbers alone. Apart from the fact, that the hash based on modular arithmetic will certainly be slower than standard hash functions, the complexity of the cryptosystem is clearly reduced with no loss of security.

As the hash depends on a user's hash key material certain requirements are essential while selecting the hash key. While it is only desirable to use strong primes for the hash modulus, it is in fact neccessary to avoid degenerated generator values that allow computations in a smooth subgroup where the discrete log problem is easy.

And finally applications that use the discrete logarithm hash must provide a reliable means to verify the integrity of the key material the hash is using, as this cannot be provided by the hash function. This does not introduce additional complexity because it implies that the hash key will be an essential component of the user's public signing key, whose integrity is a fundamental requirement of every cryptosystem in any case.

References

[And-93] Ross Anderson: The Classification of Hash Functions
In: Codes and Ciphers (proceedings of fourth IMA Conference on Cryptography and Coding, December 1093), published by IMA (1995) pp 83-93

[AV] Ross Anderson and Serge Vaudenay: Minding your p's and q's
available at :
http://www.cl.cam.ac.uk/ftp/users/rja14/psandqs.ps.gz

[Bon] Dan Boneh: Twenty Years of Attacks on the RSA Cryptosystem
In: Notices of the American Mathematical Society (AMS), Vol. 46, No. 2, 1999, pp. 203-213
http://crypto.stanford.edu/~ dabo/papers/RSA-survey.ps

[Dam-88] I.B. Damgard: Collision free hash functions and public key signature schemes
In: Advances in Cryptology, Proc. Eurocrypt 1987, LNCS 304, D.Chaum and W.L. Price, Eds., Springer-Verlag, 1988, pp. 203-216

[FSW] Niels Ferguson, Bruce Schneier and David Wagner: Security Weaknesses in Maurer-Like Randomized Stream Ciphers
http://www.counterpane.com/maurer-stream.html

[Gib-91] J.K. Gibson: Discrete logarithm hash function that is collision free and one way
In: IEE Proceedings-E, Vol. 138, No. 6, November 1991, pp. 407-410

[Pre-94] Bart Preneel: Cryptographic Hash Functions
In: European Transactions on Telecommunications, Vol. 5, No. 4, July-August 1994, pp. 431-448

[RSA-78] Ronald L. Rivest, Adi Shamir and Leonard M. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems
In: Communications of the ACM, Vol 21, February 1978, pp. 120-126

[RS-98] Ronald L. Rivest and Robert D. Silverman: Are 'Strong' Primes Needed for RSA ?
In: The 1997 RSA Laboratories Seminar Series, Seminars Proceedings, 1997
http://theory.lcs.mit.edu/ ~ rivest/
RivestSilverman-AreStrongPrimesNeededForRSA.ps

[R-2003] Ronald L. Rivest: public communication
http://www.mit.edu:8008/bloom-picayune/crypto/13190

[S-2000] Ralf Senderek: Key-Experiments - How PGP deals with manipulated keys
In: Datenschutz und Datensicherheit, Verlag Vieweg, Wiesbaden, October 2000, pp. 603-608
http://senderek.de/security/key-experiments.html