[Date Prev][Date Next][Thread Prev][Thread Next][Date Index][Thread Index]

Bleichenbacher's RSA signature forgery based on implementation error

- *From*: hal at finney.org ("Hal Finney")
- *To*: ietf-openpgp at imc.org

- *Date*: Sun, 27 Aug 2006 22:42:46 -0700 (PDT)

At the evening rump session at Crypto last week, Daniel Bleichenbacher gave a talk showing how it is possible under some circumstances to easily forge an RSA signature, so easily that it could almost be done with just pencil and paper. This depends on an implementation error, a failure to check a certain condition while verifying the RSA signature. Daniel found at least one implementation (I think it was some Java crypto code, not OpenPGP related) which had this flaw. I wanted to report on his result here so that other OpenPGP implementers can make sure they are not vulnerable. Be aware that my notes were hurried as Daniel had only a few minutes to talk.

The attack is only good against keys with exponent of 3. There are not too many of these around any more but you still run into them occasionally. It depends on an error in verifying the PKCS-1 padding of the signed hash.

An RSA signature is created in several steps. First the data to be signed is hashed. Then the hash gets a special string of bytes in ASN.1 format prepended, which indicates what hash algorithm is used. This data is then PKCS-1 padded to be the width of the RSA modulus. The PKCS-1 padding consists of a byte of 0, then 1, then a string of 0xFF bytes, then a byte of zero, then the "payload" which is the hash+ASN.1 data. Graphically:

```
00 01 FF FF FF ... FF 00  ASN.1  HASH
```

The signature verifier first applies the RSA public exponent to reveal this PKCS-1 padded data, checks and removes the PKCS-1 padding, then compares the hash with its own hash value computed over the signed data.

The error that Bleichenbacher exploits is if the implementation does not check that the hash+ASN.1 data is right-justified within the PKCS-1 padding. Some implementations apparently remove the PKCS-1 padding by looking for the high bytes of 0 and 1, then the 0xFF bytes, then the zero byte; and then they start parsing the ASN.1 data and hash. The ASN.1 data encodes the length of the hash within it, so this tells them how big the hash value is. These broken implementations go ahead and use the hash, without verifying that there is no more data after it. Failing to add this extra check makes implementations vulnerable to a signature forgery, as follows.

Daniel forges the RSA signature for an exponent of 3 by constructing a value which is a perfect cube. Then he can use its cube root as the RSA signature. He starts by putting the ASN.1+hash in the middle of

the data field instead of at the right side as it should be. Graphically:

00 01 FF FF ... FF 00 ASN.1 HASH GARBAGE

This gives him complete freedom to put anything he wants to the right of the hash. This gives him enough flexibility that he can arrange for the value to be a perfect cube.

In more detail, let D represent the numeric value of the 00 byte, the ASN.1 data, and the hash, considered as a byte string. In the case of SHA-1 this will be 36 bytes or 288 bits long. Define N as 2^288-D. We will assume that N is a multiple of 3, which can easily be arranged by slightly tweaking the message if neccessary.

Bleichenbacher uses an example of a 3072 bit key, and he will position the hash 2072 bits over from the right. This improperly padded version can be expressed numerically as 2^3057 - 2^2360 + D * 2^2072 + garbage. This is equivalent to 2^3057 - N*2^2072 + garbage. Then, it turns out that a cube root of this is simply 2^1019 - (N * 2^34 / 3), and that is a value which broken implementations accept as an RSA signature.

You can cube this mentally, remembering that the cube of (A-B) is A^3 - 3(A^2)B + 3A(B^2) - B^3. Applying that rule gives 2^3057 - N*2^2072 + (N^2 * 2^1087 / 3) - (N^3 * 2^102 / 27), and this fits the pattern above of 2^3057 - N*2^2072 + garbage. This is what Daniel means when he says that this attack is simple enough that it could be carried out by pencil and paper (except for the hash calculation itself).

Implementors should review their RSA signature verification carefully to make sure that they are not being sloppy here. Remember the maxim that in cryptography, verification checks should err on the side of thoroughness. This is no place for laxity or permissiveness.

Daniel also recommends that people stop using RSA keys with exponents of 3. Even if your own implementation is not vulnerable to this attack, there's no telling what the other guy's code may do. And he is the one relying on your signature.

Hal Finney

- Prev by Date: **Multisig (was: OpenPGP Minutes / Quick Summary)**
- Next by Date: **keys for regression testing of OpenPGP code**
- Previous by thread: **Re: OpenPGP Minutes / Quick Summary**
- Next by thread: **keys for regression testing of OpenPGP code**
- Index(es):
  - **Date**
  - **Thread**