On the *Key*-collisions in the Signature Schemes

Tomáš Rosa

ICZ a.s., Prague, CZ

Dept. of Computer Science, FEE, CTU in Prague, CZ

tomas.rosa@i.cz

Motivation to study *k*-collisions

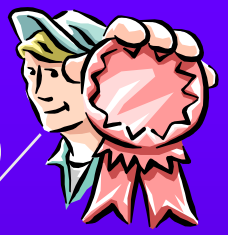
- **Def. Non-repudiation [9,10].** *The third independent party can be convinced that a particular event did (or did not) occur.*

Motivation to study *k*-collisions

- Moreover we shall require the decision made to be consistent with the objective reality.
  - It may be possible to make *just-some* decision, however we want us to be almost sure, that this decision is not only formally true. We want to be almost sure that things have happened right in that way, that we say.
- The *k*-collisions seem to be (at least theoretically) able to violate this concept.

The problem illustrated

VALID



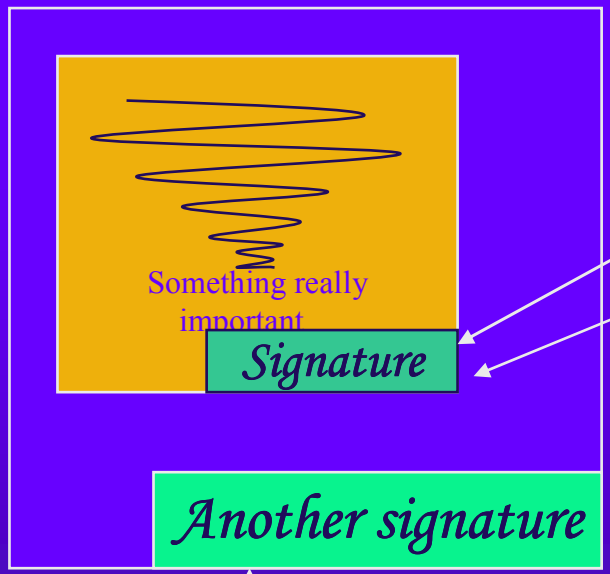
Public key A

VALID

Public key B

The third independent party tries to find out the truth...

The problem illustrated (II)



VALID

Public key A

VALID

VALID

Public key of the really honest party

Public key B

The third independent party…

The *k*-collision

- **Def. *k*-collision.** *Let (m, S) be the message m and its signature S, which is valid in the given signature scheme under the public key* Pub_A. *The pair of public keys* (Pub_A, Pub_B), $Pub_A \neq Pub_B$, *is said to be the k-collision if there is (m, S), such that S is valid signature of m under the both keys* Pub_A *and* Pub_B. *Moreover such a signature S we refer to as the k-colliding signature.*

Our contribution

- Our aim is here to show how to find the k-collision for the given message m and the k-colliding signature S for the signature schemes: RSA, DSA and ECDSA.
- The common scenario is based on the possibility of the unrestricted generation of the second colliding public key.
- There is no need for the cooperation between the holders of the public keys Pub_A and Pub_B.
  - The second user can “steal” the signature of the first one.

The *k*-collision for the RSA

- Let us have (m, S) and the public key Pub_A, $Pub_A = (n_A, e_A)$.
- We search for the second public key Pub_B, $Pub_B = (n_B, e_B)$, such that:
  - $S^{eB} \bmod n_B = r$, where $r = S^{eA} \bmod n_A$,
  - $\lceil |n_B|/8 \rceil = \lceil |n_A| /8 \rceil$.
- Such a key produces the *k*-collision for many encoding methods (mainly those in the **PKCS#1** standard).

The *k*-collision for the RSA

- We start with choosing the n_B, such that:
  - $n_B = pq$, p and q are both primes,
  - $p = 2v^{\alpha}+1$, v is a prime, $v < 2^{80}$,
  - $q = 2w^{\beta}+1$, w is a prime, $w < 2^{80}$, $w \neq v$,
  - $\lceil |n_B|/8 \rceil = \lceil |n_A| /8 \rceil$.
- Such a primes p, q induce the factor groups Z_p^* and Z_q^*, where the discrete logarithm problem (DLP) can be solved efficiently.
- Note that $gcd(p-1, q-1) = 2$.

The k-collision for the RSA

- Next we compute e_p and e_q, such that:
  - $r \equiv S^{ep} \pmod{p}$, i.e. we solve DLP in $\boldsymbol{Z}_p^*$,
  - $r \equiv S^{eq} \pmod{q}$, i.e. we solve DLP in $\boldsymbol{Z}_q^*$.
- We use the Pohlig-Hellman algorithm with the time-memory trade-off modification (see [10]) for this purpose.

The *k*-collision for the RSA

- Using the order of S in the groups Z_p^* and Z_q^* we can write:
  - $e_B \equiv e_p \pmod{ord_p(S)}$,
  - $e_B \equiv e_q \pmod{ord_q(S)}$.
- If $gcd(ord_p(S), ord_q(S)) = 1$, we can directly compute the e_B using the Chinese Remainder Theorem (CRT). In particular we use the Gauss's algorithm [10].

The *k*-collision for the RSA

- If $gcd(ord_p(S), ord_q(S)) = 2$, then we can use the CRT iff $e_p \equiv e_q \pmod 2$. Otherwise we must repeat the whole computation from the starting point.
- Once we have computed the e_B successfully, it only remains to compute the private exponent d_B from:
  - $e_B * d_B \equiv 1 \pmod{2v^{\alpha}w^{\beta}}$

The k-collision for the RSA

- The complexity of the computation is mainly given by the factors $v^{1/2}$ and $w^{1/2}$.
- The probability of success in the one pass through the algorithm described can be estimated as $63/256 \approx 25\%$ for $v, w > 2^{10}$.
  - Several passes will give the desired k-collision with the high probability.

The *k*-collision for the DSA

- **Def. DSA instance.** *The DSA instance consists of the public parameters (p, q, g), the public key y and the private key x.*
- **Def. Correct DSA instance.** *The DSA instance is said to be correct if:*
  - *p, q are both primes,*
  - $2^{L-1} < p < 2^{L}$, *for some L,* $L = 512 + 64j$, *where j is an integer,* $0 \leq j \leq 8$,
  - $2^{159} < q < 2^{160}$, $q \mid (p-1)$,
  - $g^{q} \bmod p = 1$,
  - $g^{x} \bmod p = y$.

The *k*-collision for the DSA

- Let us have (m, S), $S = (r, s)$, and the public key Pub_A, $Pub_A = y_A$, together with the public parameters (p_A, q_A, g_A).
- We search for the second public key Pub_B, $Pub_B = y_B$, together with the public parameters (p_B, q_B, g_B) and the private key x_B, such that:
  - The DSA instance $(p_B, q_B, g_B, y_B, x_B)$ is correct,
  - The public keys y_A and y_B form the *k*-collision on the *k*-colliding signature *S*.

The *k*-collision for the DSA

- We start with setting $p_B = p_A = p$, $q_B = q_A = q$.
- Let us denote $h = SHA\text{-}1(m)$.
- We compute:
  - $\alpha = g_A^{wh} y_A^{wr} \bmod p$, where $w*s \equiv 1 \pmod q$.
  - Note that $\alpha = g_A^{kA} \bmod p$, where k_A is the unknown "message key" (see [6]).
  - Note that $\alpha \bmod q = r$.
- Next we choose an integer k_B, $1 < k_B < q$, and compute z as $z*k_B \equiv 1 \pmod q$.
  - Note that we keep the k_B secret.

The *k*-collision for the DSA

- Let $g_B = \alpha^z \bmod p$.
  - Note that $(g_B^{kB} \bmod p) \bmod q = r$.
  - The k_B becomes the "message key" for the user B.
- Finally we compute:
  - $x_B = t(k_B s - h) \bmod q$, where $r*t \equiv 1 \pmod q$,
  - $y_B = g_B^{xB} \bmod p$.
- It can be easily shown that:
  - The DSA instance $(p_B, q_B, g_B, y_B, x_B)$ is correct,
  - The public keys y_A, y_B form the *k*-collision on the *k*-colliding signature *S* (with the high probability).

The *k*-collision for the DSA

- Let us denote $\beta \equiv k_A k_B^{-1} \pmod q$ and note that then $g_B = g_A^{\beta} \bmod p$.
- We can write:
  - $x_B \equiv ht(\beta^{-1} - 1) + \beta^{-1} x_A \pmod q$, where $r*t \equiv 1 \pmod q$,
  - $y_B \equiv g_A^{ht(1-\beta)} y_A \pmod p$.
- Note that the user A doesn't know k_B, while the user B doesn't know k_A.
  - The private keys are properly separated, unless the whole DSA can be broken.

The *k*-collision for the ECDSA

- We've used the very general algebraic properties of the DSA.
- These properties are the same as those of the ECDSA.
- From here it follows, that our attack on the DSA can be routinely extended on the ECDSA.
  - It is elaborated in the original paper.

Tamper Resistant Key Generation

- It is the strong and general countermeasure against attacks based on *k*-collisions.
- Motto: *Nobody (including the legitimate holder of the private key!) should have the chance to generate the keys with the values chosen completely at her/his will.*
  - So far there are almost no restrictions for the legitimate users and from here follows the threat of *k*-collisions.

Tamper Resistant Key Generation

- The concept of the *Tamper Resistant Key Generation (TRKG)* involves the two main primitive procedures:
  - ***GenKey:*** *(SEED)* $\rightarrow$ *(PublicKey, PrivateKey, PublicParameters,* ***Witness****)*,
  - ***VerifyKey:*** *(PublicParameters, PublicKey,* ***Witness****)* $\rightarrow$ *(“OK“/“FALSE“).*

Tamper Resistant Key Generation

- The main characteristics of the *GenKey* are:
  - It generates cryptographically strong keys,
  - It is one-way with the respect to the input parameters,
  - Any modification, which would lead to the violation of the previous conditions, is detectable by the *VerifyKey* procedure.

Tamper Resistant Key Generation for the (EC)DSA

- It can be based on the existing concept of the “certificates of SEEDs” [6], however this concept must be extended also on the generators of the working subgroups.
  - Our attacks show that the standard FIPS PUB 186-2 should be revised in this way.

Tamper Resistant Key Generation for the RSA

- It seems to be an open hard problem, since:
  - The *Witness* should be a public value, which:
    - Allows us to detect any discrepancies in the key generation procedure,
    - Doesn't reveal any sensitive information about the prime factors of the RSA modulus.
- On the other hand it seems to be also problem to find the method producing the k-collision for the restricted value of e_B:
  - $e_B < 2^{32}$,
  - or it seems to be even harder if $e_B =^{!} F4$.

Temporary countermeasures

- Include the public key certificate (or the detailed *key info*) in the data to be signed.
- Restrict the value of the RSA public exponent, for example $e < 2^{32}$, or even $e =^{!} F4$. It should not decrease the cryptographic strength if we use the proper formatting methods (otherwise there can be some attacks, see [10]).
- Force the users to use the service of some honest key generation authority (e.g. the service offered by the certification authority).

Conclusion

- We have presented the notion of *k*-collisions and we have shown how this concept can be used for attacks on the service of non-repudiation.
- Effective methods for the construction of *k*-collisions for the RSA and (EC)DSA schemes were elaborated.
  - We have shown that none of these schemes provides the non-repudiation service in itself.
- We have identified the concept of the *Tamper Resistant Key Generation* as the strong general countermeasure against these attacks.
  - The non-repudiation of signatures extend transitively to the non-repudiation of the fair key generation.

Questions…

…are welcome.

Thank you…

…for your attention.

Literature

1. Anderson, R. and Needham, R.: Robustness Principles for Public Key Protocols, *in Proc. of CRYPTO '95*, pp. 236-247, Springer-Verlag, 1995.
2. Anderson, R. and Vaudenay, S.: Minding your p's and g's, *in Proc. of ASIACRYPT '96*, pp. 26-35, Springer-Verlag, 1996.
3. Chen, M. and Hughes, E.: Protocol Failures Related to Order of Encryption and Signature: Computation of Discrete Logarithms in RSA Groups, in *Proc. of ACISP '98*, pp. 238-249, Springer-Verlag, 1998.
4. Desmedt, Y., Landrock, P., Lenstra, A., McCurley, K., Odlyzko, A., Rueppel, R. and Smid, M.: The Eurocrypt '92 Controversial Issue - Trapdoor Primes and Moduli, in *Proc. of EUROCRYPT '92*, pp. 194-199, Springer-Verlag, 1992.
5. FIPS PUB 180-1: *Secure Hash Standard (DSS)*, National Institute of Standards and Technology, January 2001, http://www.itl.nist.gov/fipspubs/fip180-1.htm.
6. FIPS PUB 186-2: *Digital Signature Standard (DSS)*, National Institute of Standards and Technology, January 2001, http://csrc.nist.gov/publications/fips/fips186-2/fips186-2.pdf.
7. Gordon, D.-M.: Designing and Detecting Trapdoors for Discrete Log Cryptosystems, in *Proc. of CRYPTO '92*, pp. 66-75, Springer-Verlag, 1992.
8. Johnson, D., Menezes, A. and Vanstone, S.: The Elliptic Curve Digital Signature Algorithm (ECDSA), *International Journal of Information Security*, Vol 1, Issue 1, pp. 36-63, Springer-Verlag, 2001.
9. Landwehr, C.-E.: Computer Security, *International Journal of Information Security*, Vol 1, Issue 1, pp. 3-13, Springer-Verlag, 2001.
10. Menezes, A.-J., van Oorschot, P.-C. and Vanstone, S.-A.: *Handbook of Applied Cryptography*, CRC Press, 1996, online at http://www.cacr.math.uwaterloo.ca/hac/.
11. Microsoft: CryptoAPI Version 2.0, *MSDN - Platform SDK*, 2000.
12. *PKCS#1 v2.1: RSA Cryptography Standard*, RSA Laboratories, DRAFT2, January 5 2001.
13. *Public-Key Cryptography Standards (PKCS)*, RSA Security, available at http://www.rsasecurity.com/rsalabs/pkcs/index.html.
14. Pohlig S.-C., Hellman M.-E.: An improved algorithm for computing logarithms over GF(p) and its cryptographic significance, *IEEE Transactions on Information Theory*, 24 (1978), 106-110.
15. Rivest, R.-L., Shamir, A. and Adleman L.: A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, pp. 120-126, 1978.
16. Vaudenay, S.: Hidden Collisions on DSS, *in Proc. of CRYPTO '96*, pp. 83-88, Springer-Verlag, 1996.
17. *Vyhláška č. 366/2001 Sb.*, ÚOOÚ 2001.
18. Zákon o elektronickém podpisu, *zákon č. 227/2000 Sb.*, 2000.