# Key-collisions in (EC)DSA: Attacking Non-repudiation*

Tomáš Rosa

http://crypto.hyperlink.cz, email: t_rosa@volny.cz

**Abstract.** A new kind of attack on the non-repudiation property of digital signature schemes is presented. We introduce a notion of key-collisions, which may allow an attacker to claim that the message (presented to a judge) has been signed by someone else. We show how to compute key-collisions for the DSA and ECDSA signature schemes effectively. The main idea of these attacks has been inspired by the well-known notion of message-collisions, where an attacker claims that the signature presented at the court belongs to a different message. Both of these collision-based attacks significantly weaken the non-repudiation property of signature schemes. Moreover, they weaken the non-repudiation of protocols based on these schemes. It is shown that key-collision resistance of the (EC)DSA schemes requires the incorporation of a mechanism ensuring honest generation of (EC)DSA instances. The usage of such a mechanism shall be verifiable by an independent third party without revealing any secret information. We propose and discuss basic general countermeasures against key-collision attacks on the (EC)DSA schemes.

## 1. Introduction

The service of non-repudiation is one of the most basic cryptographic goals ([11]). The commonly agreed definition of this service says that: *The non-repudiation property of a given action allows an independent third party to make sure that a particular event did (or did not) occur* ([10]). Note that the independent party is typically a judge. Such a property is of a great importance for applications where cryptographic mechanisms enter an area of law. A good example of such a service is the introduction of *electronic signature standards and laws* ([3, 4]), which is an ongoing activity through the whole world. There are also other areas where the non-repudiation plays an important role, for instance we may refer to asymmetric traitor tracing schemes [9], which achieve their non-repudiation by using signature schemes having this property. All these examples show that contemporary cryptographic mechanisms must not only protect data, but also be judiciously sound.

In this article we show an approach which can be used to break the non-repudiation service in systems which are based on the (EC)DSA ([6]) signature schemes. The main idea behind our attack may be called an *"alternative explanation"*. This is a familiar and effective way in which a signature may be denied. An attacker constructs the alternative explanation that argues that there is the (mathematically) valid signature, while she claims that she has never signed the document presented at the court. Such an argument is then presented to the judge. We should note, that the provable mathematical connection between document signing and signature verification is that proper signing implies verifying the signature as a valid one. Proving this implication in the opposite direction, i.e. that verifying the signature as a valid one implies that the document was properly signed before, is somewhat tricky. In most schemes, this "proof" is just based on that there is no sound alternative explanation of why the verification procedure says that the signature is valid. Then, presenting such an alternative explanation means that the trial must be tried basing on other evidences, what may at least be very inconvenient. Therefore, a possibility of the alternative explanation existence should be minimized, especially on the elementary mathematical level. A well-known way to get the alternative explanation is to exploit collisions of a hash function used in the signature scheme. The attacker then claims that she has signed a *different message*, instead of that message presented at the court. Such a threat is well recognized and an adequate attention is usually paid to collision resistance of hash functions in signature schemes. Apparently, some attention is also paid to the inner collision resistance of the signature transformations ([15, 16]). In this article we show that it is also possible to get a collision of public keys (more precisely of public signature scheme instances), which we refer to as a key-collision (*k*-

* Extended version of the paper supporting a brief talk given at the CRYPTO 2002 Rump Session.

collision, c.f. §2 for the definition). Then, the alternative explanation says that it was *someone else* who has signed that message. A straightforward decision that users have *both* signed this message would not be proper, likewise we do not accept the decision that the user has signed *both* messages in the case of the message-collision (*m*-collision). Moreover, there may be cases where it is logically impossible for two users to have signed the given message. For instance, when the signature scheme may be employed in an authentication service which disallows more than one user to be authenticated at the same time.

It is easy to see that *m*-collisions and *k*-collisions both weaken the non-repudiation property according to the definition stated above. It is worth to note again (c.f. [15]) that attacks on the non-repudiation may expose surprising weaknesses, since it is often a private key owner who plays the role of an adversary here. However, we usually tend to view such a person as the honest user who is a prime target of malicious attackers, and therefore she must be protected using all accessible means.

In the following text we show a formal definition of the term *k*-collision (§2), then we introduce the notion of GDSA (§3), which is a general construction that encapsulates elementary common algebraic properties of DSA and ECDSA. Besides allowing us to study our subject for both schemes at the same time, it also demonstrates the generality of properties which allow us to search for *k*-collisions effectively. A concrete algorithm for *k*-collision searching is presented in §4 then. Section §5 shows basic limits of *k*-collision computation for (EC)DSA. Use of this observation is made in §6, where general countermeasures are introduced. Some heuristic remarks that did not fit elsewhere are summarized in §7 as possible inspirations for further research. Finally, we conclude in §8.

**Proposition 1.1.** *Unless stated otherwise, the term "breaking a signature scheme (instance)" means an ability to compute a private key only from publicly known values (i.e. a public key, message signatures, etc.).*

**Proposition 1.2.** *Unless stated otherwise, the elements of $\boldsymbol{Z}_n$ will be regarded as the lowest positive integers representing particular equivalence classes in $\boldsymbol{Z}_n$.*

**Proposition 1.3.** *In case it is clear from a context, we will shorten a notation of indexed variables according to the following example: $x_A, x_B \rightarrow_{written\ as}\ x_{A,B}$.*

## 2. *K*-collisions – Definitions

**Definition 2.1.** *We write as $(Pub, m, S)_\rho$ the relation $\rho$ saying that S is a valid signature of the message m under the public signature scheme instance Pub.*

The term *public instance* and the structure of *S* are defined in §3. It suffices to understand them heuristically through a general meaning here.

**Definition 2.2 (*k*-collision).** *The five-tuple $(Pub_A, m_A, Pub_B, m_B, S)$, where $Pub_A$ and $Pub_B$ are public instances of a signature scheme, $m_A$ and $m_B$ are messages and S is a signature, form a k-collision if the following holds: $Pub_A \neq Pub_B$, $(Pub_A, m_A, S)_\rho$ and $(Pub_B, m_B, S)_\rho$. Furthermore the public instances $Pub_{A,B}$, messages $m_{A,B}$, and their signature S are referred to as k-colliding public instances, k-colliding messages and k-colliding signature respectively.*

**Definition 2.3 (1st and 2nd order *k*-collision).** *We use the term $1^{st}$ order k-collision to refer to the k-collision, where $m_A = m_B$. We use the term $2^{nd}$ order k-collision to refer to the k-collision, where $m_A \neq m_B$.*

The sets of all 1st order and 2nd order *k*-collisions decompose the set of all *k*-collisions for a given signature scheme. In this paper, we study algorithms which can be used for the purpose of *k*-collision searching. The following definition introduces the basic properties of such algorithms.

**Definition 2.4 (*k*-collision searching algorithms properties).**

*i)* *We say that an algorithm is non-cooperative if it finds a k-collision for a given public signature scheme instance $Pub_A$, a k-colliding message $m_A$, and a signature S, such that $(Pub_A, m_A, S)_\rho$, without needing any further information which is not publicly accessible. We say that the algorithm is cooperative, otherwise.*

*ii)* *An algorithm for k-collision searching is message-independent if it allows the k-colliding messages $m_{A,B}$ to be pre-set to arbitrary strings.*

*iii)* *An algorithm is proper if it does not disallow respective owners of k-colliding public instances $Pub_{A,B}$ to know their relevant private keys. I.e., we may assume that the owner of $Pub_A$ knows $Priv_A$ and the owner of $Pub_B$ knows $Priv_B$.*

Non-cooperative algorithms have a special importance here, since they allow an attacker to do so-called *signature stealing*. Using such an algorithm, the attacker may pretend to have signed an important document which has actually been signed by someone else. The real author of the signature does not have to provide any secret information to the attacker, so the attacker "steals" her signature. There are practical cases in which such an attack brings some benefits to the attacker, for instance, bypassing an authorship proving service.

A message-independent algorithm for *k*-collision searching can be used to search for both 1st order and 2nd order *k*-collisions. Furthermore, it is reasonable to require such an algorithm to be proper in case the judge requests the respective owners of $Pub_{A,B}$ to prove that they know their respective private keys. Therefore, we focus on the design of a proper message-independent non-cooperative algorithm for *k*-collision searching. Such an algorithm is described in §4.

### 2.1 Illustrative example of a practical attack

Let us imagine the following scenario: There is a scientific conference whose potential participants are requested to submit anonymous papers. To thwart cheating, each researcher must also append a string of her digital signature of the paper. Of course, only the signature is appended, not a public key. I.e. using the above-mentioned notation, researcher *A* sends $paper_A$ and *S* from a triplet $(Pub_A, paper_A, S)_\rho$. When the paper is accepted and or a disputation occurs, the researcher must prove that she owns the proper verification/signing keys, i.e. that *S* is her proper signature. This should prevent changing the author and or the content of the paper later on. However, it would not work, when it is possible to construct *k*-collisions. Fro instance, having a *k*-collision $(Pub_A, paper_A, Pub_B, paper_B, S)$, such that $paper_A = paper_B$, user *B* may pretend to be the author of the paper. Moreover, if it is possible to compute the *k*-collision non-cooperatively, then user *B* can do so without an agreement with user *A*.

What follows is that we cannot take the signature string *S* in itself as a fingerprint of the document content ($paper_A$) and the identity of the signatory ($Pub_A$). Only the full triplet $(Pub_A, paper_A, S)_\rho$ can be used for such a purpose. Without having investigated the area of *k*-collisions, such a conclusion would not be so obvious and applications, as the one described here, might easily be deployed. (Author would like to note here that such an application was already met in practice.)

### 2.2 Another example

In the European Electronic Signature Standardization Initiative [3], there is a term *advanced electronic signature* which is defined in the following way: *electronic signature which meets the following requirements:* ***a) it is uniquely linked to the signatory****; b) it is capable of identifying the signatory; c) it is created using means that the signatory can maintain under his sole control; and d) it is linked to the data to which it relates in such a manner that any subsequent change of the data is detectable (see Directive 1999/93/EC)*. It is the point (a) which is very important for us here. Let us have a cryptographic digital signature scheme which is planned to be used for a construction of a particular advanced electronic signature scheme. Note that this is the kind of electronic signature which is commonly understood as a "safe" one through the whole European Union, and therefore almost all applications tend to achieve at least this "level" of an electronic signature scheme. It is also accepted that a general cryptographic digital signature scheme is automatically also an advanced electronic signature scheme. However, if the digital signature scheme used allows an attacker to construct *k*-collisions, then the electronic signature scheme based on it clearly cannot be called advanced, since the condition in (a) would not be fulfilled. Such a discrepancy can probably lead to tough judicial consequences, and therefore *k*-collisions should not be underestimated here.

## 3. Generalized DSA

The main purpose of Generalized DSA (GDSA) is to generalize common algebraic properties of DSA [6] and ECDSA [6, 8]. It is introduced here solely for the purpose of developing a general model for dealing with *k*-collisions, which have to give valid results for both DSA and ECDSA. Therefore, the particular security requirements for GDSA based signature schemes, other than those being connected with *k*-collision attacks, are not discussed here. Moreover, we make an effort to keep the GDSA definition algebraically close to the way in which the (EC)DSA schemes are practically realized (here we may differ from the approaches generalizing (EC)DSA from other viewpoints, e.g. [1, 15]).

**Definition 3.1 (Generalized DSA – GDSA).** *A GDSA instance consists of public parameters, a private key, a public key, and public transformations.*

- *The public parameters are represented as the three-tuple (**P**, n, g), where **P** is a cyclic group, g is a generator of a prime-order subgroup **G** of **P** and n is the order of **G**. Unless stated otherwise, the group operation on **P** will be written in multiplicative notation and the identity element of **P** will be denoted as **id**.*
- *The private key (x) is an integer satisfying $0 < x < n$. The public key (y) is an element of **P**, $y = g^x$.*
- *The public transformations consist of two publicly known mappings denoted as H and φ. H is a hash function, $H: \mathbf{M} \rightarrow H(\mathbf{M})$, where **M** is a set of input messages to be processed. We assume that H implicitly incorporates a string-to-integer conversion, i.e. $H(\mathbf{M}) \subset \mathbf{Z}$.*
- *The purpose of φ is to define a conversion function $\varphi: \mathbf{P} \rightarrow \mathbf{Z}_n$. Note that since n is a prime, $\mathbf{Z}_n$ is isomorphic to **GF**(n). Our reasoning doesn't depend on the concrete definition of φ here.*
- *We denote the GDSA instance as Inst, Inst = (**P**, n, g, x, y, H, φ). The GDSA instance without the private key x will be referred to as the public GDSA instance (or the public part of GDSA instance) Pub, Pub = (**P**, n, g, y, H, φ).*

**Definition 3.2 (Proper GDSA instance).** *Let (**P**, n, g, x, y, H, φ) be a GDSA instance. This instance is said to be proper if the following conditions hold:*

*i)* ***P** has a proper structure – the meaning of this condition depends on the particular kind of GDSA (e.g. **P** must not be built over a weak elliptic curve, n is large enough, etc.)*
*ii)* *n is a prime, n divides the order of **P***
*iii)* *ord(g) = n*
*iv)* *$0 < x < n$*
*v)* *$g^x = y$*

The purpose of this definition is to summarize general requirements to which a GDSA instance must conform. We will use this definition when showing that a particular generated instance is sound enough to be regarded as a properly working scheme without any obvious marks of an attack.

**Definition 3.3 (GDSA signing algorithm).** *Let (**P**, n, g, x, y, H, φ) be a GDSA instance and let m be a message to be signed. The signing operation then proceeds with the following steps:*

*i)* *compute the integer h, h = H(m)*
*ii)* *generate a secret random integer k, $0 < k < n$; note that k will be referred to as a nonce*
*iii)* *compute the integer r, $r = \varphi(g^k)$*
*iv)* *compute the integer s, $s = (h + rx)k^{-1} \bmod n$, where $kk^{-1} \equiv 1 \pmod n$*
*v)* *if either r = 0 or s = 0, repeat the whole computation from (ii)*
*vi)* *the pair (r, s) is the signature of m*

**Definition 3.4 (GDSA verifying algorithm).** *Let Pub, Pub = (**P**, n, g, y, H, φ), be a public GDSA instance and let m be a message, whose signature (r, s) has to be verified. The verifying operation then proceeds with the following steps:*

*i)* *if either r = 0 or s = 0, then the signature is rejected as an invalid one*
*ii)* *compute the integer h, h = H(m)*

*iii)* *compute the integer $u_1$, $u_1 = h*s^{-1} \bmod n$, where $ss^{-1} \equiv 1 \pmod n$*

*iv)* *compute the integer $u_2$, $u_2 = r*s^{-1} \bmod n$*

*v)* *compute the integer $r'$, $r' = \varphi(g^{u1}y^{u2})$*

*vi)* *the signature is valid iff $r' = r$, i.e. $(Pub, m, (r,s))_\rho$ iff $r' = r$*

### 3.1 DSA

The DSA scheme [6] will be viewed as the GDSA scheme where $\boldsymbol{P}$ is a multiplicative group $\boldsymbol{Z}_p^*$, where $p$ is a prime and $n|(p-1)$. The identity element is $\boldsymbol{id} = [1]_p$. The conversion function $\varphi$ is defined as the mapping $\varphi: \boldsymbol{Z}_p^* \rightarrow \boldsymbol{Z}_n: a \rightarrow ((a \bmod p) \bmod n)$. The standard [6] currently prescribes the use of the SHA-1 [5] as $H$. Note that the order of the working prime order subgroup is often ([6]) denoted as $q$, therefore we adopt this notation in Appendix A.

### 3.2 ECDSA

The ECDSA scheme [6, 8] will be viewed as the GDSA scheme where $\boldsymbol{P} = E(\boldsymbol{F}_q)$, $\boldsymbol{F}_q$ is a finite field isomorphic to $\boldsymbol{GF}(q)$ and $E$ is a suitable planar elliptic curve over $\boldsymbol{F}_q$. $E(\boldsymbol{F}_q)$ is an abelian group of points on the curve $E$ (together with the special *point at infinity* denoted as $O$). The order of $E(\boldsymbol{F}_q)$ is denoted $\#E(\boldsymbol{F}_q)$, so $n|\#E(\boldsymbol{F}_q)$. The identity element is $\boldsymbol{id} = O$. The group operation on $E(\boldsymbol{F}_q)$ is written in additive notation, where a $v$-times iterated addition of a point $A$, $A \in E(\boldsymbol{F}_q)$, is denoted as $B = [v]A$, $B \in E(\boldsymbol{F}_q)$. The conversion function $\varphi$ is the mapping $\varphi: E(\boldsymbol{F}_q) \rightarrow \boldsymbol{Z}_n: A = (x, y) \rightarrow \text{int}(x) \bmod n$, where int($x$) is an integer representation for the $x$-coordinate of $A$, $x \in \boldsymbol{F}_q$. The concrete definition of int(.) depends on the way in which the field $\boldsymbol{F}_q$ is constructed. The standard [6] currently prescribes the use of the SHA-1 [5] as $H$.

## 4. *K*-collisions for GDSA

In this part we show how to effectively compute $k$-collisions for the GDSA scheme. The algorithm introduced here is message-independent, non-cooperative, and proper. Therefore, it can be also used for the purpose of signature stealing (c.f. §2).

Theoretically, it might be possible for $k$-colliding instances to belong to different kinds of GDSA. However, our algorithm presented here is based on that the instances belong to the same kind of GDSA, i.e. they are both either DSA instances or ECDSA instances. This assumption allows us to design the algorithm in an effective way, while it does not bring any practical restriction.

**Algorithm 4.1 (Computing a *k*-collision for GDSA).**

Input:

- Public GDSA instance $Pub_A = (\boldsymbol{P}_A, n_A, g_A, y_A, H_A, \varphi_A)$, which is a public part of a proper GDSA instance $Inst_A$.
- Message $m_A$ and its signature $(r, s)$, such that $(Pub_A, m_A, (r, s))_\rho$.
- Message $m_B$, which the $k$-collision is computed for.

Output:

- GDSA instance $Inst_B = (\boldsymbol{P}_B, n_B, g_B, x_B, y_B, H_B, \varphi_B)$.
- $k$-collision $(Pub_A, m_A, Pub_B, m_B, (r, s))$.

Computation:

i) place $\boldsymbol{P} = \boldsymbol{P}_B = \boldsymbol{P}_A$, $n = n_B = n_A$, $H = H_B = H_A$ and $\varphi = \varphi_B = \varphi_A$

ii) compute the integer $h_A$, $h_A = H(m_A)$

iii) compute the integer $u_1$, $u_1 = h_A*s^{-1} \bmod n$, where $ss^{-1} \equiv 1 \pmod n$

iv) compute the integer $u_2$, $u_2 = r*s^{-1} \bmod n$

v) compute $\alpha$, $\alpha \in \boldsymbol{P}$, $\alpha = g_A^{u1}y_A^{u2}$

vi) generate a secret random integer $z$, $0 < z < n$

vii) compute the integer $k_B$, such that $zk_B \equiv 1 \pmod n$, i.e. $k_B \equiv z^{-1} \pmod n$

viii) compute the integer $h_B$, $h_B = H(m_B)$

ix) if $k_B s - h_B \equiv 0 \pmod n$ goto (vi)
x) set $x_B = (k_B s - h_B) r^{-1} \bmod n$, where $r r^{-1} \equiv 1 \pmod n$
xi) set $g_B = \alpha^z$
xii) set $y_B = g_B^{xB}$
xiii) if $y_B = y_A$ goto (vi)
xiv) set $Inst_B = (\boldsymbol{P}_B, n_B, g_B, x_B, y_B, H_B, \varphi_B)$, $Pub_B = (\boldsymbol{P}_B, n_B, g_B, y_B, H_B, \varphi_B)$
xv) return $Inst_B$, $(Pub_A, m_A, Pub_B, m_B, (r, s))$

■

Note that setting $\boldsymbol{P}_B = \boldsymbol{P}_A$ (and $n_B = n_A$) should not look suspicious later on (e.g. at the court), because DSA and ECDSA schemes were both developed with the possibility of sharing the group $\boldsymbol{P}$ and its prime order subgroup among many independent users. Sometimes, it is even recommended in the case of ECDSA to use prescribed elliptic curves rather than generating new ones (c.f. curves in [6]).

The value of $z$ generated in step (vi) should be discarded after finishing the computation, since it may allow other attackers to discover the private key $x_B$ and it would serve as an easy proof of a "cooked" generation of $Inst_B$.

Theorems on general properties of the GDSA instance $Inst_B$ produced by algorithm 4.1, together with the main theorem stating that the algorithm produces a $k$-collision, follow. Unless stated otherwise, these theorems and lemmas presume the validity of the input assumptions of 4.1, and their symbols refer to its input, temporary and output variables. Owners of the instances $Inst_A$ and $Inst_B$ are referred to as the users $A$ and $B$ respectively.

**Lemma 4.2 (Tractability of algorithm 4.1).** *Algorithm 4.1 is tractable if the signing and verifying algorithms of the particular GDSA are tractable.*

*Proof.* This algorithm uses the same kind of operations as the signing and verifying algorithms (c.f. definitions 3.3 and 3.4).

■

**Lemma 4.3 (On the generator $g_B$).** *For $g_B$ it holds that $g_B = (g_A^{kA})^z$, where $k_A$ is the nonce used by the user A for the (r, s) computation. Furthermore, we have $g_B^{kB} = g_A^{kA}$ and $\varphi(g_B^{kB}) = \varphi(g_A^{kA}) = r$.*

*Proof.* The equation $g_B = (g_A^{kA})^z$ follows immediately from step (xi) of algorithm 4.1 and the assumption of $(Pub_A, m_A, (r, s))_\rho$. Also following are $g_B^{kB} = g_A^{kA}$ and $r = \varphi(g_B^{kB})$, since $z k_B \equiv 1 \pmod n$ and $n$ is the order of $g_A$.

■

**Lemma 4.4 (On the values of $x_{A,B}$ and $y_{A,B}$).** *It holds that:*
*i)* $x_B = [(k_A^{-1} k_B h_A - h_B) r^{-1} + k_A^{-1} k_B x_A] \bmod n$
*ii)* $x_A = [(k_A k_B^{-1} h_B - h_A) r^{-1} + k_A k_B^{-1} x_B] \bmod n$
*iii)* $y_B = g_A^{\gamma}$, *for* $\gamma \equiv (h_A - k_A k_B^{-1} h_B) r^{-1} + x_A \pmod n$
*iv)* $y_A = g_B^{\lambda}$, *for* $\lambda \equiv (h_B - k_A^{-1} k_B h_A) r^{-1} + x_B \pmod n$

*Proof.*

i) According to step (x) of algorithm 4.1, we have $x_B = (k_B s - h_B) r^{-1} \bmod n$. We assume that $(r, s)$ is a valid signature of $m_A$ computed according to definition 3.3. From here we get $s = (h_A + r x_A) k_A^{-1} \bmod n$, where $k_A k_A^{-1} \equiv 1 \pmod n$ and $k_A$ is a random integer, $0 < k_A < n$. Substituting this expression into the equation for $x_B$ we have $x_B = [(k_A^{-1} k_B h_A - h_B) r^{-1} + k_A^{-1} k_B x_A] \bmod n$.

ii) Follows directly from (i).

iii) Using (i), lemma 4.3 and the equation from step (xii) of 4.1 we get the equation for $y_B$.

iv) Analogically as (iii).

■

**Theorem 4.5 (On termination).** *Algorithm 4.1 terminates.*

*Proof.* There are only two loops in 4.1 and these are in steps (ix) and (xiii). The loop in step (ix) acts if and only if $k_B s - h_B \equiv 0 \pmod n$. Since $\gcd(s, n) = 1$, there is exactly one value of $k_B$, $0 < k_B < n$, satisfying this congruence. Therefore, an infinite loop does not occur here for randomly chosen values of $k_B$. The second loop is in step (xiii) and it acts if and only if $y_B = y_A$. Using lemma 4.4 we can rewrite this condition as $y_A =$

$g_A^{\gamma}$, for $\gamma \equiv (h_A - k_A k_B^{-1} h_B) r^{-1} + x_A \pmod{n}$, where $k_A$ is a random number, $0 < k_A < n$, which is fixed for all loops through algorithm 4.1. This condition holds if and only if $(h_A - k_A k_B^{-1} h_B) r^{-1} \equiv 0 \pmod{n}$. Again, there is only one value of $k_B$ which satisfies this congruence. Therefore, this loop is finite for randomly chosen values of $k_B$.

■

**Theorem 4.6 (Properness of $Inst_B$).** *The GDSA instance $Inst_B$ computed by algorithm 4.1 is proper.*

*Proof.* Let us check the conditions (i - v) required by definition 3.2:

- i-ii) These conditions are fulfilled according to step (i) of algorithm 4.1. We rely on the assumption that $Inst_A$ is proper, and therefore $\boldsymbol{P}_A$ must have a proper structure, and $n$ must divide the order of $\boldsymbol{P}_A$.
- iii) Recall that $n = n_B = n_A$ and lemma 4.3 above. Since we assume that the instance $Inst_A$ is proper, it holds that $\mathrm{ord}(g_A) = n$, where $n$ is a prime. Because of $n$ being a prime and $k_A$, $z < n$ it follows that $\gcd(n, k_A z) = 1$. Therefore $\mathrm{ord}(g_B) = \mathrm{ord}(g_A) = n$.
- iv) Since $x_B$ is a result of operation modulo $n$ it trivially holds that $0 \le x_B < n$. It remains to check that $x_B \neq 0$. Let us suppose that $x_B = 0$. Since $\gcd(r^{-1}, n) = 1$ and $x_B = (k_B s - h_B) r^{-1} \bmod n$, this equation holds if and only if $k_B s - h_B \equiv 0 \pmod{n}$. However, this is prevented by step (ix). Therefore $0 < x_B < n$.
- v) It follows directly from step (xii) of algorithm 4.1.

■

**Theorem 4.7 (Algorithm 4.1 produces a *k*-collision).** *The five-tuple ($Pub_A$, $m_A$, $Pub_B$, $m_B$, $(r, s)$) computed by algorithm 4.1, is a k-collision. Furthermore, the algorithm is a message-independent non-cooperative and proper one.*

*Proof.* We assume that $(Pub_A, m_A, (r, s))_{\rho}$. It remains to show that also $(Pub_B, m_B, (r, s))_{\rho}$ and $Pub_A \neq Pub_B$. We use verifying algorithm 3.4 for $m_B$ and $Pub_B$ at first. In the steps (i - iii) of 3.4 we obtain $u_1 = h_B s^{-1} \bmod n$, $u_2 = r s^{-1} \bmod n$, where $s s^{-1} \equiv 1 \pmod{n}$. In step (v) of 3.4 we get $r' = \varphi(g_B^{u1} y_B^{u2}) = \varphi(g_B^{\omega})$, where $\omega \equiv h_B s^{-1} + x_B r s^{-1} \pmod{n}$. From step (x) of 4.1 we have $x_B = (k_B s - h_B) r^{-1} \bmod n$. Substituting this value to the congruence for $\omega$ we get $\omega \equiv h_B s^{-1} + k_B - h_B s^{-1} \equiv k_B \pmod{n}$, so $r' = \varphi(g_B^{kB})$. According to lemma 4.3 we get $r' = r$. Therefore, the signature $(r, s)$ is valid. Furthermore, according to the condition in step (xiii) of 4.1, the public instances $Pub_A$ and $Pub_B$ differ at least in the values of public keys $y_A$ and $y_B$, therefore, ($Pub_A$, $m_A$, $Pub_B$, $m_B$, $(r, s)$) is a $k$-collision. Moreover, there is neither a restriction for messages $m_{A,B}$ nor a need for a cooperation with the user $A$. Therefore, this algorithm is message-independent and non-cooperative. Since the validity of the $A$'s private key is left intact and the $B$'s private key is computed in step (x) of 4.1, this algorithm is also proper.

■

**Theorem 4.8 (On attacker's private key secrecy).** *Unless the particular realization of GDSA can be broken (c.f. proposition 1.1), there is a negligible probability that the user A is able to break B's instance $Inst_B$.*

*Proof.* If the particular GDSA realization (e.g. DSA, or ECDSA) cannot be broken, then there is no way for an ordinary user to break someone else's instance. However, we shall check if the special construction of $Inst_B$ used in algorithm 4.1 helps the users $A$ and $B$ to break each other's instances or not. At first, we observe that the users $A$ and $B$ play symmetric roles in our scheme: Both of them may regard her GDSA instance as the first one for which the second user has computed her $k$-colliding instance. This symmetry can be seen from lemma 4.4. Therefore, if there is a way for the user $A$ to discover the $B$'s private key, then there is also a way for the user $B$ to discover $A$'s private key. Because there is no need for a cooperation between $A$ and $B$, it follows that the user $B$ could break the $A$'s GDSA instance simply by computing an appropriate $k$-collision. If we assume that breaking the particular GDSA scheme is hard, then there must be a negligible probability that the appropriate $k$-collision would be found using algorithm 4.1. Therefore, the construction used in 4.1 cannot practically allow the user $B$ to break the $A$'s instance. From the symmetry observed above, it follows that the construction does not practically allow the user $A$ to break $B$'s instance.

■

## 5. Basic Limits for General *K*-collision Searching Algorithms

In §4 we have seen an effective algorithm for *k*-collision searching. However, there are other approaches to this problem. The aim of this paragraph is to show basic limits for general *k*-collision searching algorithms in GDSA schemes. It will help us when designing appropriate general countermeasures in §6.

**Definition 5.1.** *Let us denote $\varphi^{|\boldsymbol{G}}$ the GDSA (c.f. def. 3.1) conversion function $\varphi$ restricted on **G**, where **G** is the prime-order subgroup of **P** generated by g.*

It has been shown (c.f. [1, 12]), that $\varphi^{|\boldsymbol{G}}$ is an almost-bijective mapping. Moreover, we conjecture the following hypothesis for secure GDSA instances.

**Hypothesis 5.2 (Inner collision resistance).** *For randomly chosen pairs $(g_1, g_2)$, such that $g_{1,2} \in \boldsymbol{P}$, $ord(g_1) = ord(g_2) = n$, there is no tractable algorithm producing integers v, w, such that $\varphi^{|\boldsymbol{G}}(g_1^{\,v}) = \varphi^{|\boldsymbol{G}}(g_2^{\,w})$, while $g_1^{\,v} \neq g_2^{\,w}$.*

**Theorem 5.3 (Limiting theorem).** *Unless GDSA schemes can be broken, there is no feasible proper k-collision searching algorithm producing k-collision $(Pub_A, m_A, Pub_B, m_B, (r, s))$ for $Pub_{A,B}$, such that $Pub_A = (\boldsymbol{P}, n, g_A, y_A, H, \varphi)$ and $Pub_B = (\boldsymbol{P}, n, g_B, y_B, H, \varphi)$, where $g_{A,B}$ are arbitrary given generators.*

*Proof (assuming validity of 5.2).* Obviously, the GDSA instance $Inst = (\boldsymbol{P}, n, g, x, y, H, \varphi)$ can be broken if we can solve the discrete logarithm problem (DLP) according to the generator $g$ effectively. We show how a general proper *k*-collision searching algorithm can be used for solving the DLP. Let us assume that we want to solve the DLP on **P** with the base $g$ for the public key $y$ to get the private key $x$. In the first step, we choose temporary helping GDSA instances $Inst_A$ and $Inst_B$, such that $Inst_A = (\boldsymbol{P}, n, g, x_A, y_A, H, \varphi)$, $Inst_B = (\boldsymbol{P}, n, y, x_B, y_B, H, \varphi)$, where $x_{A,B}$ are arbitrarily chosen private keys and $y_{A,B}$ are the appropriate public keys. Note that $Inst_B$ uses the public key $y$ in the place of its generator $g_B$. Also, note that $x_{A,B}$ may be fixed later on, during the *k*-collision searching process. We only assume that these values are known then (i.e. the algorithm is proper). Now we start searching to find two *k*-colliding messages $m_{A,B}$, and a *k*-colliding signature $(r, s)$. Since we know $x_{A,B}$ then, we can compute particular nonces $k_{A,B}$ for $m_{A,B}$ and $(r, s)$ easily. It holds that $r = \varphi^{|\boldsymbol{G}}(g^{kA}) = \varphi^{|\boldsymbol{G}}(y^{kB})$. Using the properties of $\varphi^{|\boldsymbol{G}}$ stated above, we can rewrite this as $g^{kA} = y^{kB}$ with a high probability. Since $n$ is the order of $g$ and $y = g^x$, we have $k_A \equiv xk_B \pmod{n}$. From here we can compute the value of $x$ easily, which means that our attack on $x$ is finished.

Hypothesis 5.2 tells us that the general algorithm used above cannot rely on inner collisions of $\varphi^{|\boldsymbol{G}}$. Although there may still be singular problems with these collisions, they should appear as often as if $\varphi^{|\boldsymbol{G}}$ was a pseudorandom function. Such rare exceptions may then be easily overcome by repeating the whole process several times, using different values of $g_{A,j}$ and $g_{B,j}$ in each pass $j$. For instance, we can set $g_{A,j} = g^j$, $g_{B,j} = y^j$ for $j \in \langle 1, n)$. This construction leads to $jk_{A,j} \equiv jxk_{B,j} \pmod{n}$ then, so still $k_{A,j} \equiv xk_{B,j} \pmod{n}$.

■

## 6. Countermeasures

### 6.1 Basic Reasoning

Theorem 5.3 tells us that even if we place no restrictions on the public and private keys (except that we want to know these respective keys) and set no rules for the *k*-colliding messages and the signature, the *k*-colliding instances $Pub_{A,B}$ still cannot have their remaining public parameters set to arbitrary values. Their choices must in some way be dependent to allow an attacker to compute a *k*-collision. It requires further research to say how far this dependency goes, but we can conclude heuristically that the dependency is probably stronger than the necessity to omit the setup used in the premise of theorem 5.3.

**Corollary 6.1 (Hypothesis on dependency).** *If the particular GDSA scheme cannot be broken, then there is no proper k-collision searching algorithm that allows the public parameters of k-colliding instances to be chosen independently.*

### 6.2 Online Protocol

Basing on corollary 6.1, we propose the idea of a simple but strong countermeasure: All public parameters for GDSA schemes must be *independently* generated by a trusted third party *on a per-instance basis*. The authority shall also issue a certificate of proper GDSA instance generation. For example, the scenario under which users generate their GDSA instances and requests for a public key certificate should be like this one:

i. **user -> certification authority:** *certification_request_start*
ii. **certification authority -> user:** *public_ parameters_(**P**, n, g)*, *token*
iii. user chooses a private key and computes the public key
iv. **user -> certification authority:** *public_key*, *possession_proof*, *token*

We assume the conversion function $\varphi$ to be implicitly set according to the particular kind of GDSA. The token introduced in steps (ii) and (iv) helps the certification authority ensure that the particular public parameters generated are used only by one user. The possession_proof introduced in step (iv) serves as a proof of private key possession. It has to ensure that whatever *k*-collision searching algorithm should be used, it must be a proper one (c.f. definition 2.4 and corollary 6.1). The public key certificate issued according to the protocol written above also serves as the certificate of a proper GDSA instance generation.

Note that in the case of ECDSA, this setup may be attacked by the inner *m*-collisions presented in [15, §4.2 – Duplicate Signatures]. However, disclosing such an *m*-collision leads to the private key disclosure and therefore it is not considered a security weakness ([15]). If we assume that the certification authority is honest, then the inner *m*-collisions of DSA described in [16] are successfully defeated by this setup.

The proposed protocol may also be used in a situation when users need to share a common group $\boldsymbol{P}$ and its prime-order subgroup (e.g. they all want to use the same standardized elliptic curve). In such a case, the authority independently generates for each request a new value of the generator *g* only, while the remaining public parameters ($\boldsymbol{P}$, *n*) stay constant. The impossibility of *k*-collision computing then follows directly from theorem 5.3.

### 6.3 Non-invertible GDSA Setup

It might be useful to substitute the above-mentioned protocol by a non-invertible verifiable GDSA setup. Such a setup may be regarded as an extension of the DSA and ECDSA setups currently defined in [6, 8]. One idea could be to extend the concept of "seeded" prime number (DSA case) searching or elliptic curve (ECDSA case) construction to also cover the subgroup generator *g*. This item remains unprotected by those mechanisms currently and it has already been shown ([16]) that it is no good. The value of SEED used to initialize the setup would then serve as the certificate of proper instance generation. This improving step would help to reduce the risk of *k*-collisions prominently. However, there still remains a possibility of generating a 2nd order *k*-collision cooperatively (the main problem here is that users are not forced to choose their public parameters independently). Let $Inst_A = (\boldsymbol{P}, n, g, x_A, y_A, H, \varphi)$ be a GDSA instance generated properly (e.g. certified by the SEED) and let $(m_A, m_B)$ be two different messages for which users *A* and *B* want to find a *k*-collision. Now, the user *A* computes the signature $(r, s)$ of $m_A$ in $Inst_A$ normally, while she keeps the value of the nonce *k*. This value is passed to the user *B* then, together with the signature $(r, s)$ and the public part of $Inst_A$. The user *B* then constructs $Inst_B = (\boldsymbol{P}, n, g, x_B, y_B, H, \varphi)$, where the private key $x_B$ is computed from the congruence $s \equiv (H(m_B) + x_B r)k^{-1} \pmod{n}$. It is easy to see that $(Pub_A, m_A, Pub_B, m_B, (r, s))$ is the 2nd order *k*-collision. The drawback of this process is that it requires the cooperation between the users who then know each other's private keys. Moreover, it allows a third party seeing this *k*-collision to discover the linear relation between $x_{A,B}$ as $x_A - x_B \equiv r^{-1}(H(m_B) - H(m_A)) \pmod{n}$. On the other hand, it clearly breaks the countermeasure based on the simple setup extension. It follows that there is a need for a broader extension of the current standard – it shall also cover the generation of the private key. Moreover, the certificate of the proper private key generation shall be verifiable, without disclosing any secret information about the key. It remains an open research question on how to do such an extension securely.

### 6.4 Notary Services and-or Authentication of Public Instances

Another kind of countermeasure may be deployed in systems which use some kind of notary services (for overview on notary services see [11]). It may be feasible in such a situation to require every signed document to be over-signed by a trusted third party. The signature should cover: the message, its (primary) signature, and the public part of the GDSA instance which shall be used for the (primary) signature verification. Similar techniques may also be based on a time stamping service.

It might be also tempting to propose the following countermeasure: request users to sign not only the message, but also the public part of GDSA instance. The signature should then cover the string $m||public\_instance$ instead of plain $m$. However, this countermeasure seems to have several weaknesses, at least from a theoretical point of view. It still leaves a possibility for an attacker to claim that, due to an error, the *public_instance* was appended badly or even it was not appended at all. Furthermore, there is a threat of cooperative 2nd order $k$-collisions which should be investigated to devise an acceptable proof of security. So far, it is known that it is not enough to hash $m$ together with the public key only. The whole set of public parameters must be added, too. There is still an open question of how far this countermeasure is affected by the properties of the conversion function $\varphi$, especially by its invertibility. Moreover, this countermeasure affects data formats and the behavior of all client applications, in contrast with the protocol proposed in §6.2 above, which only needs the extension of the certificate request process.

## 7. Closing Remarks

We stress that $k$-collisions are not only a problem for DSA and ECDSA. The same problem may be studied in the RSA ([13]), ElGamal ([2]) and Schnorr ([14]) signature schemes (as well as in the others). Of course, there is a difference in how feasible and conspicuous these attacks are. Feasibility says whether it is possible to do such an attack, while the second criterion tells us if it is easy to recognize marks of the attack later. The current state of research shows that the DSA and ECDSA schemes belong to the group where $k$-collision attacks are feasible and do not leave special marks on $k$-colliding instances. ElGamal and Schnorr schemes probably belong to the same group, since they share those general algebraic properties which were used for our attack. However, both of them introduce certain properties which induce limitations. For example, the algorithm presented here can produce 1st order $k$-collisions only, when applied (and adjusted) on Schnorr scheme (due to binding of the message $m$ being signed and the value of $r$, $r = g^k \bmod p$, as $e = H(m||r)$, where the value of $e$ becomes a part of the signature, c.f. [14, p. 168]). Therefore, we may conclude that these schemes come with some (maybe planned or unplanned) built-in countermeasures which are (however) not strong enough to defeat all these attacks. On the other hand, RSA seems to belong to the group where these attacks are still feasible, but they *do* produce special marks which could be used by a third party to even break one of the $k$-colliding instances.

## 8. Conclusion

We have introduced the notion of key-collisions in signature schemes, which may be regarded as a kind of attack parallel to well known message-collisions. Both of them share the common idea of an *alternative explanation* saying why the judge has a message and its (mathematically) valid signature when the user swears that she did not sign it. Instead of claiming that it was a *different message* from what has been signed in reality, an attack based on key-collisions leads to claiming that it was a *different user* who has signed the message or even who has signed a different message. Next, we presented a (trivially) feasible algorithm for key-collision searching in DSA and ECDSA, which is message-independent and does not require any cooperation between the owners of colliding instances. Therefore, an attacker may use this algorithm to steal a signature of another user. The effectiveness of the algorithm comes mainly from the ability of the attacker to generate (EC)DSA instances with a relatively high degree of freedom. We showed that even the legitimate owner of the key should not have the ability to generate her key completely at her will without having to be able to present a proof of its honest creation.

It was shown in §5, that unless DSA and ECDSA schemes can be broken, possibilities for key-collision searching in these respective schemes can be prevented. We have developed a general countermeasure based on the simple online procedure for a key generation (§6.2). When deployed in an existing information system, it only changes a certificate request process. Other processes and data structures remain unchanged. Several other possible countermeasures were proposed and discussed, too (§6.3, §6.4).

## Acknowledgements

The author is grateful to Vlastimil Klima and Jared Smolens for their helpful comments on drafts of this paper.

## References

[1] D. R. L. Brown. Generic Groups, Collision Resistance, and ECDSA, February 2002, IEEE 1363, [7].
[2] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory*,Vol. 31, pp. 469–472, IEEE, 1985.
[3] EESSI. The European Electronic Signature Standardization Initiative. See http://www.ict.etsi.org/eessi/EESSI-homepage.htm.
[4] E-SIGN. The Electronic Signatures in Global and National Commerce Act, enacted on June 30, 2000. See http://www.cybercrime.gov/esign.htm.
[5] FIPS PUB 180-1. Secure Hash Standard (SHA-1), National Institute of Standards and Technology, January 2001.
[6] FIPS PUB 186-2. Digital Signature Standard (DSS), National Institute of Standards and Technology, January 27, 2000, update: October 5, 2001.
[7] IEEE P1363. Standard Specifications for Public Key Cryptography, August 1998. See http://grouper.ieee.org/groups/1363.
[8] D. Johnson, A. J. Menezes and S. A. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA), *International Journal of Information Security*, Vol 1, Issue 1, pp. 36-63, Springer-Verlag, 2001.
[9] A. Kiayias and M. Yung. Breaking and Repairing Asymmetric Public-Key Traitor Tracing, to appear in *Proc. of the 2002 ACM Workshop on Digital Rights Management*.
[10] C. E. Landwehr, Computer Security, *International Journal of Information Security*, Vol 1, Issue 1, pp. 3-13, Springer-Verlag, 2001.
[11] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1996.
[12] P. Q. Nguyen and I.E. Shparlinski. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces, *Journal of Cryptology*, Vol. 15, 3/2002, pp. 151-176, Springer-Verlag, 2002.
[13] R. L. Rivest, A. Shamir and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, pp. 120-126, 1978.
[14] C. P. Schnorr. Efficient Signature Generation by Smart Cards, *Journal of Cryptology*, Vol. 4, pp. 161–174, Springer-Verlag, 1991.
[15] J. Stern, D. Pointcheval, J. Malone-Lee and N. P. Smart. Flaws in Applying Proof Methodologies to Signature Schemes, in *Proc. of CRYPTO 2002*, LNCS 2442, pp. 93-110, Springer-Verlag, 2002.
[16] S. Vaudenay. Hidden Collisions on DSS, in *Proc. of CRYPTO '96*, pp. 83-88, Springer-Verlag, 1996.

## Appendix A: Algorithm 4.1 edited for DSA and ECDSA

The algorithm for effective *k*-collision searching (c.f. §4) edited for the DSA and ECDSA schemes is presented here. It should be a helpful illustration of how the general reasoning done for the GDSA model (c.f. §3, §4) transforms back to these particular schemes.

Notes: The conversion function $\varphi$ together with the hash function *H* (SHA-1) are implicitly defined for the DSA and ECDSA schemes (c.f. §3.1, §3.2). Therefore we omit them from the notation of the (EC)DSA instances here. Furthermore, we use the prime *p* alone when referring to the multiplicative group $\boldsymbol{P} = \boldsymbol{Z}_p^*$ in the case of DSA.

**Algorithm DSA-4.1 (Computing a *k*-collision for DSA).**

Input:

- Public DSA instance $Pub_A = (p_A, q_A, g_A, y_A)$.
- Message $m_A$ and its signature $(r, s)$, such that $(Pub_A, m_A, (r, s))_\rho$.
- Message $m_B$, which the *k*-collision is computed for.

Output:

- DSA instance $Inst_B = (p_B, q_B, g_B, x_B, y_B)$.
- *k*-collision $(Pub_A, m_A, Pub_B, m_B, (r, s))$.

Computation:

i) place $p = p_B = p_A$, $q = q_B = q_A$

ii) compute the integer $h_A$, $h_A = \text{SHA-1}(m_A)$

iii) compute the integer $u_1$, $u_1 = h_A * s^{-1} \bmod q$, where $ss^{-1} \equiv 1 \pmod q$

iv) compute the integer $u_2$, $u_2 = r * s^{-1} \bmod q$

v) compute $\alpha$, $\alpha = g_A^{u1} y_A^{u2} \bmod p$

vi) generate a secret random integer $z$, $0 < z < q$

vii) compute the integer $k_B$, $zk_B \equiv 1 \pmod q$, i.e. $k_B \equiv z^{-1} \pmod q$

viii) compute the integer $h_B$, $h_B = \text{SHA-1}(m_B)$

ix) if $(k_B s - h_B) \equiv 0 \pmod q$ goto (vi)

x) set $x_B = (k_B s - h_B) r^{-1} \bmod q$, where $rr^{-1} \equiv 1 \pmod q$

xi) set $g_B = \alpha^z \bmod p$

xii) set $y_B = g_B^{xB} \bmod p$

xiii) if $y_B = y_A$ goto (vi)

xiv) set $Inst_B = (p_B, q_B, g_B, x_B, y_B)$, $Pub_B = (p_B, q_B, g_B, y_B)$

xv) return $Inst_B$, $(Pub_A, m_A, Pub_B, m_B, (r, s))$

■

**Algorithm ECDSA-4.1 (Computing a *k*-collision for ECDSA).**

Input:

- Public ECDSA instance $Pub_A = (E(\boldsymbol{F}_q)_A, n_A, G_A, Y_A)$.
- Message $m_A$ and its signature $(r, s)$, such that $(Pub_A, m_A, (r, s))_\rho$.
- Message $m_B$, which the $k$-collision is computed for.

Output:

- ECDSA instance $Inst_B = (E(\boldsymbol{F}_q)_B, n_B, G_B, x_B, Y_B)$.
- $k$-collision $(Pub_A, m_A, Pub_B, m_B, (r, s))$.

Computation:

i) place $E(\boldsymbol{F}_q)_B = E(\boldsymbol{F}_q)_A$, $n = n_B = n_A$

ii) compute the integer $h_A$, $h_A$ = SHA-1($m_A$)

iii) compute the integer $u_1$, $u_1 = h_A * s^{-1} \bmod n$, where $ss^{-1} \equiv 1 \pmod n$

iv) compute the integer $u_2$, $u_2 = r * s^{-1} \bmod n$

v) compute $\alpha$, $\alpha = [u_1]G_A + [u_2]Y_A$

vi) generate a secret random integer $z$, $0 < z < n$

vii) compute the integer $k_B$, $zk_B \equiv 1 \pmod n$ i.e. $k_B \equiv z^{-1} \pmod n$

viii) compute the integer $h_B$, $h_B$ = SHA-1($m_B$)

ix) if $(k_B s - h_B) \equiv 0 \pmod n$ goto (vi)

x) set $x_B = (k_B s - h_B) r^{-1} \bmod n$, where $rr^{-1} \equiv 1 \pmod n$

xi) set $G_B = [z]\alpha$

xii) set $Y_B = [x_B]G_B$

xiii) if $Y_B = Y_A$ goto (vi)

xiv) set $Inst_B = (E(\boldsymbol{F}_q)_B, n_B, G_B, x_B, Y_B)$, $Pub_B = (E(\boldsymbol{F}_q)_B, n_B, G_B, Y_B)$

xv) return $Inst_B$, $(Pub_A, m_A, Pub_B, m_B, (r, s))$

■