Multi-user Schnorr security, revisited

Daniel J. Bernstein[1,2]

[1] Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607–7045, USA
djb@cr.yp.to

[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. Three recent proposals for standardization of next-generation ECC signatures have included "key prefixing" modifications to Schnorr's signature system. Bernstein, Duif, Lange, Schwabe, and Yang stated in 2011 that key prefixing is "an inexpensive way to alleviate concerns that several public keys could be attacked simultaneously".

However, a 2002 theorem by Galbraith, Malone-Lee, and Smart states that, for the classic Schnorr signature system, single-key security tightly implies multi-key security. Struik and then Hamburg, citing this theorem, argued that key prefixing was unnecessary for multi-user security and should not be standardized.

This paper identifies an error in the 2002 proof, and an apparently insurmountable obstacle to the claimed theorem. The proof idea does, however, lead to a different theorem, stating that single-key security of the classic Schnorr signature system tightly implies multi-key security of the key-prefixed variant of the system. This produces exactly the opposite conclusion regarding standardization.

Keywords: Schnorr signatures, multi-user security, proof errors

1 Introduction

For certain types of discrete logarithm based systems, which we dub "Schnorr-like" signature schemes, we have shown that the security does not decrease with the number of users.
—Galbraith–Malone-Lee–Smart, 2002 [16]

For Schnorr and ECDSA type schemes, one does not need to include the public key in the signing process, since security in the multi-user setting is roughly the same as in the single-user setting ... —Struik, 2015 [35], citing [16]

This work was supported by the National Science Foundation under grant 1018836 and by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005. Permanent ID of this document: 75659a3df50fb5d53ccb4c2a0bb8ec83e7464420. Date: 2015.10.12.

Most of the literature on signature-system security focuses on the problem of forging messages under *one* targeted public key. However, the real-world attacker actually sees *many* public keys, and presumably is happy if he solves the problem of forging messages under any one of those keys. (For example, he wants to take over *somebody's* account to steal money, gain computer power, relay more attacks, etc.) If the attacker has a noticeable probability of solving this problem, then the signature system is protecting *most* users but not *all* users, and it is difficult to argue that the signature system should be considered to be secure.

An attacker who solves the first problem, namely forging messages under one targeted public key, can also solve the second problem in the same time and with the same success probability: the attacker simply targets the first key and ignores the remaining keys. The multi-key problem is therefore no *harder* than the single-key problem.

What about the converse? Could the multi-key problem be *easier* than the single-key problem? Perhaps switching from the single-key problem to the multi-key problem gives the attacker an advantage, compared to the original problem.

For comparison, it is well known that for many secret-key protocols, such as typical message-authentication systems, the attacker *does* gain a factor N in switching from the single-key problem to the N-key problem; see, e.g., [**4**] and [**13**]. This is one of the reasons that I recommend upgrading from 128-bit secret keys to 256-bit secret keys. Similarly, if an attack against 256-bit elliptic curves were discovered with cost only $2^{128}/\sqrt{N}$ to break 1 of N signature keys for large N, then I would recommend upgrading from 256-bit elliptic curves to larger elliptic curves.[3]

Fortunately, for all of the signature systems considered in this paper, there are no known multi-key attacks faster than single-key attacks. But does the absence of any *known* speedup justify assuming that no speedup exists? Perhaps there is a gap between the problems that cryptanalysts have been studying, such as *single-key* security of 256-bit elliptic-curve signatures, and the problems that actually matter for cryptographic users, such as *multi-key* security of 256-bit elliptic-curve signatures. One can reasonably be worried about the *possibility* of multi-key attacks, and about the lack of study of this possibility.

1.1. Previous work. This issue was directly tackled 13 years ago in a short paper [**16**] by Galbraith, Malone-Lee, and Smart (henceforth "GMLS"). That paper has two main results.

The first result states that, for any signature system, attacking N keys at once cannot increase the attacker's success probability by a factor above N. This has an easy proof, which works as follows.

Say we have an N-key attack that, with probability p, successfully produces a forgery. Consider the following 1-key attack:

[3] For comparison, I am not at all concerned about the fact that well-known batch discrete-logarithm algorithms (see, e.g., [**15**], [**28**], [**21**], and [**11**]) cost "only" $2^{128}\sqrt{N}$ to break *all* N of the 256-bit signature keys. The attacker cannot afford $2^{128}\sqrt{N}$; the attacker cannot even afford 2^{128}; if cost is limited to, e.g., 2^{100} then these algorithms have negligible chance of finding *any* discrete logarithms.

- Generate a list of $N-1$ new keys, by applying the standard key-generation procedure $N-1$ times.
- Insert the target key into the list at a random position.
- Run the N-key attack.
- When the N-key attack asks for a signature under the target key, apply the target signing oracle.
- When the N-key attack asks for a signature under any of the $N-1$ new keys, apply the standard signing procedure.
- If the N-key attack successfully produces a forgery, and this forgery is under the target key, then output the forgery.

The cost of the 1-key attack is practically identical to the cost of the N-key attack. The success probability of the 1-key attack cannot be smaller[4] than p/N. In other words, the success probability of the N-key attack cannot be larger than N times the success probability of the 1-key attack.

Unfortunately, this first result is unsatisfactory, precisely because it is not tight. One can easily imagine N being 2^{30} or even larger; increasing the attacker's success probability by a factor 2^{30} could convert a minor concern into a critical real-world threat.

The second GMLS result states that, for the Schnorr signature system and more generally for "Schnorr-like" signature systems, attacking N keys at once provides no advantage at all. The proof is not as obvious and is analyzed in detail later in this paper.

This second result is completely satisfactory, eliminating any possibility of the aforementioned gap. The limitation to "Schnorr-like" signature systems might annoy theoreticians aiming for more generality, but practitioners seem happy with these signature systems.

1.2. Standardization. Internet standards, such as the TLS standard used to secure HTTPS connections, are maintained by the Internet Engineering Task Force (IETF). IETF delegates research questions to its research arm, the Internet Research Task Force (IRTF), and in particular delegates cryptographic questions to part of IRTF, the Crypto Forum Research Group (CFRG).

For the last 18 months, about 3000 messages, the primary topic on the CFRG mailing list has been elliptic-curve cryptography. For the last 6 months, about 700 messages, the primary topic has been elliptic-curve signatures. There have been five specific proposals of signature systems and several "tweaks" of those proposals.

The proposals vary in several choices, including "key prefixing", explained in Section 2. Three of the five signature proposals to CFRG use key prefixing, while two do not. All of the key-prefixed proposals appear to stem from a 2011 paper

[4] The GMLS paper claims that the probability is *exactly* p/N. In fact, the probability could be larger: if one of the new keys happens to collide with the target key, then a forgery under that new key will also be output as a successful forgery from this algorithm, so there are 2 positions out of N that work rather than just 1 out of N. What matters for the first result is that the probability is *at least* p/N.

[9] by Bernstein, Duif, Lange, Schwabe, and Yang specifying the "Ed25519" signature system. That paper does not present or cite security proofs related to key prefixing, but nevertheless includes key prefixing as "an inexpensive way to alleviate concerns that several public keys could be attacked simultaneously".

In a CFRG message dated 4 September 2015, Struik [35] recommended against key prefixing. "Ideally, signing should be possible without requiring the signer to access its public key", he wrote; and, finally bringing security proofs into the discussion, he cited the GMLS paper to conclude that "one does not need to include the public key in the signing process". Hamburg [20] echoed Struik's recommendation, saying that "there is little gain" to key prefixing "in light of the papers he cited".

This story is obviously a triumph for provable security: a tight standard-model security theorem is used to simplify and streamline cryptography that is being standardized for real-world applications. Some designers guessed that key prefixing would be helpful for security; but the choice of a standard is guided by theorems rather than by guesses. The GMLS theorem shows that the intended security goal is achieved without this unnecessary design element, so the design element is eliminated, the same way that HMQV [27] "provably dispenses" with some elements of MQV.

1.3. Contributions of this paper. This paper identifies an error in the GMLS proof, and an apparently insurmountable obstacle to the claimed theorem.

On the positive side, this paper shows that the attacker's probability of breaking any 1 of N signature keys *in the key-prefixed variant* of the Schnorr system is at most the attacker's probability of breaking a targeted signature key in the classic Schnorr system. In other words, anyone who believes that the classic Schnorr system is secure in the single-key case also has to believe that the key-prefixed variant of the system is secure in the multi-key case.

If the theorem claimed by GMLS were proven then it would trivially imply this new theorem, by two different proofs, where one proof moves vertically and then horizontally in the following diagram, and the other proof moves horizontally and then vertically:

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{multi-key security of}\\ \text{classic Schnorr}\end{array}} \arrow[r, Rightarrow, "\text{easy}"] & \boxed{\begin{array}{c}\text{multi-key security of}\\ \text{key-prefixed Schnorr}\end{array}} \\
\boxed{\begin{array}{c}\text{single-key security of}\\ \text{classic Schnorr}\end{array}} \arrow[u, Rightarrow, "\text{GMLS claim}"] \arrow[r, Rightarrow, "\text{easy}"'] \arrow[ur, Rightarrow, bend left, "\begin{array}{c}\text{new}\\ \text{theorem}\end{array}"'] & \boxed{\begin{array}{c}\text{single-key security of}\\ \text{key-prefixed Schnorr}\end{array}} \arrow[u, Rightarrow, "\text{GMLS claim}"']
\end{tikzcd}$$

However, the vertical implications in this diagram are actually unproven and presumably unprovable, leaving the new diagonal theorem as the only known way to move from single-key security to multi-key security.

What does this mean for key prefixing, assuming single-key security for the classic Schnorr system? The new theorem says that key prefixing guarantees multi-key security. The obstacle to proving the originally claimed theorem says that without key prefixing there is no guarantee of multi-key security. The conclusion is clear: one should use key prefixing.

See Section 2 for definitions of the signature systems; Section 3 for the new theorem; Section 4 for analysis of the GMLS error; and Section 5 for generalizations.

1.4. Standardization, revisited. After the announcement of this new theorem (and of the obstacle to the originally claimed theorem), CFRG settled on a signature system that includes key prefixing.

One can still characterize this story as a triumph for provable security: cryptography equipped with a tight standard-model security theorem is standardized for real-world applications, in preference to slightly more streamlined cryptography that is not equipped with such a theorem. But what is particularly interesting about the complete story is the reversal of recommendations: one moment provable security is being used to justify a recommendation *against* key prefixing, and the next moment provable security is being used to justify a recommendation *for* key prefixing.

In retrospect the first recommendation can and should be dismissed as not being a valid example of provable security. There was a small error in the proof; this error led to an erroneous claim of a theorem, which in turn led to an unjustified recommendation regarding standardization. But this avalanche of errors raises a much larger question: what protection does the cryptographic standardization process have against errors in provable-security claims?

For comparison, the security impact of errors in *programs* is one of the central topics in the security literature. The unfortunate reality is that user security can be, and frequently is, compromised by accidental (or malicious) programming errors anywhere in a very large trusted code base. Simply fixing each bug as it is discovered obviously does not produce a secure system, whereas there is at least some hope of obtaining a secure system through systematically (1) reducing the amount of trusted code and (2) eliminating errors in the code that remains. See, e.g., [5].

There is similarly widespread awareness that some provable-security claims have errors,[5] but the primary response consists of fixing each error as it is discovered. There has been relatively little effort to understand and systematically mitigate the security impact of a continuing series of errors.

A small portion of the provable-security literature has been formalized and verified by proof-checking tools; see, e.g., [1]. Koblitz in [24] and [25] disputed

[5] For example, a famous 2001 paper of Shoup [33] pointed out, and partially patched, an error in a provable-security claim that had been published seven years earlier for the "OAEP" system by Bellare and Rogaway [3]. Koblitz and Menezes have maintained a decade-long drumbeat of highlighting errors and limitations in provable-security claims; their series of "Another look at provable security" papers [26] is, as far as I know, the only serious survey of this quagmire.

the value of such tools, and in particular wrote that the tools are "limited to generating proofs for trivial steps"; but the GMLS error was inside a trivial step, and I think that formally verifying the GMLS proof *would* have caught this error. One can view this as an argument that standardization committees should disregard provable-security claims that have not been formally verified. A counterargument is that formal verification is expensive, probably infeasible to carry out at the required scale, although focusing standardization efforts on a smaller number of cryptographic constructions might make it feasible.

In [6] I argued, for other reasons, that security proofs should not be advertised to users selecting cryptographic systems, although they are sometimes "a useful guide to cryptanalysts". With this approach, a standardization committee deciding between the original Schnorr system and the key-prefixed variant would ask cryptanalysts about the safety of the two systems. Part of the job of the cryptanalysts is to consider multi-key security, and it's easy to see that key prefixing blocks some potential attack strategies against multiple keys, so if there isn't literature exploring the limits of those attack strategies then cryptanalysts will naturally recommend key prefixing as the conservative choice. It's possible that the GMLS paper would have convinced cryptanalysts that key prefixing isn't useful, but my experience is that cryptanalysts are skeptical of security proofs and are unlikely to rely on them without first verifying them; there is an important difference in incentives between provers and cryptanalysts. Similarly, the proof in *this* paper would have to survive the cryptanalyst filter before it could influence the standardization process.

Perhaps cryptanalysts will actually take the time to study multi-key Schnorr and conclude that it's just as safe as single-key Schnorr, whether or not there's a security proof. In other words, it could turn out that the GMLS error is hard, or impossible, for an attacker to exploit. This is analogous to the well-known fact that an error in the trusted code base is not necessarily a security hole; many errors are hard, or impossible, to exploit. On the other hand, it could also turn out that multi-key Schnorr really does lose security. The broader point is that blindly trusting claims of provable security is obviously not a safe way to select cryptographic standards.

2 A generalization of the Schnorr signature system

This section presents a parametrized family of signature systems that includes (1) the classic Schnorr signature system and (2) the key-prefixed variant of the system.

2.1. Signature systems. A signature system, by definition, consists of four sets and three algorithms satisfying one condition. The four sets are a set $\{k\}$ of "secret keys", a set $\{K\}$ of "public keys", a set $\{M\}$ of "messages", and a set $\{\Sigma\}$ of "signatures". The three algorithms are as follows:

- Gen ("key generation"). Inputs: none. Outputs: a public key and a secret key.

- Sign ("signing"). Inputs: a message and a secret key. Output: a signature.
- Verify ("verification"), required to be deterministic. Inputs: a message, a signature, and a public key. Output: True or False.

The condition is that Verify(M, Σ, K) = True whenever Gen() $= (K, k)$ and Sign$(M, k) = \Sigma$; i.e., verification accepts any signature produced by signing, under any keys produced by key generation.

Beware that several variations of this definition appear in the literature. Sometimes Sign is allowed to maintain state, i.e., to output a new secret key that is then used as a replacement for the original secret key; see, e.g., the original Goldwasser–Micali–Rivest [19] definition of signature systems, or more recent papers on hash-based signatures. Sometimes Sign is allowed to fail after a specified number of signatures. Sometimes Sign and Verify are allowed to abort, begging the question of what a user is supposed to do for messages that trigger the abort. Sometimes Gen takes a "security parameter" as input. Sometimes cost limits are placed on Gen, Sign, and Verify as part of the definition, rather than as part of subsequent cost analyses.

2.2. Invalid inputs. In typical formalizations of algorithms, inputs and outputs are required to be strings, so the sets above are also required to be sets of strings. People defining signature systems conventionally assume that the inputs are in the specified sets. This leaves unspecified what the algorithms do if inputs are "invalid", i.e., not in the specified sets.

Invalid inputs can trigger security failures even for algorithms that are believed to be secure for valid inputs. Obviously one can build artificial attack examples where, e.g., providing a non-message as input convinces Sign to reveal its secret key, or providing a non-signature as input convinces Verify to output True no matter what the message is. For non-artificial attack examples on other protocols see, e.g., [12] and [29]. The signature systems in this paper allow all strings as messages, so there is no issue for Sign in these systems, but the specifications of Verify include explicit checks for invalid inputs.

The usual arguments for assuming input validity are that (1) this shortens the algorithm statements and (2) implementors can (for every commonly used set) easily add checks for valid inputs. The main counterargument is that this hides from the reader an algorithm component that often has an impact on speed, simplicity, and most importantly security; omitting critical checks from algorithm statements is encouraging the implementor to shoot himself in the foot. See [22] for a recent attack using invalid inputs to steal secret keys from two widely deployed ECDH implementations.

2.3. Classic Schnorr signatures. Fix a positive integer g; a prime number ℓ; a group $G \subseteq \{0,1\}^g$ of size ℓ, written additively (binary operation $+$, neutral element 0, inversion $-$); a nonzero element B of G; and a deterministic algorithm $H : \{0,1\}^* \to \mathbf{Z}/\ell$. Here $\mathbf{Z}/\ell$ is the ring of integers modulo ℓ. Define $s = \lceil \log_2 \ell \rceil$, and for each $k \in \mathbf{Z}/\ell$ define $\overline{k} \in \{0,1\}^s$ as the s-bit little-endian representation of the unique representative of k in $\{0, 1, \ldots, \ell - 1\}$.

Note that G is self-delimiting (i.e., the concatenation RM for $R \in G$ and $M \in \{0,1\}^*$ determines the pair (R, M)), since it consists entirely of g-bit strings. Similarly, the set $\{\overline{k} : k \in \mathbf{Z}/\ell\}$ is self-delimiting.

The **classic Schnorr signature system with group G, base point B, and hash function H**, denoted $\text{Schnorr}_{G,B,H}$, is defined as follows. Note that the other parameters g, ℓ above are determined by G and thus do not need to be included in the notation: the elements of G (e.g., 0) are g-bit strings for a unique g, and ℓ is the number of elements of G.

The set of secret keys is $\{\overline{k} : k \in \mathbf{Z}/\ell\}$. The set of public keys is G. The set of messages is $\{0,1\}^*$. The set of signatures is $\{0,1\}^{2s}$. Gen() works as follows:

- Generate a uniform random $k \in \mathbf{Z}/\ell$.
- Compute $K = kB$.
- Return $(K, \overline{k})$.

Sign$(M, \overline{k})$ for $k \in \mathbf{Z}/\ell$ works as follows:

- Generate a uniform random $r \in \mathbf{Z}/\ell$.
- Compute $R = rB$.
- Compute $C = H(RM)$.
- Compute $S = r + Ck$. (Note that $SB = rB + CkB = R + CK$, since $\ell B = 0$.)
- Return $\overline{C}\,\overline{S}$.

Verify(M, Σ, K) works as follows:

- Find the unique $(C, S) \in (\mathbf{Z}/\ell)^2$ such that $\Sigma = \overline{C}\,\overline{S}$. If no such (C, S) exists, return False.
- If $K \notin G$, return False.
- Compute $R = SB - CK$.
- If $C \neq H(RM)$, return False.
- Return True.

If Gen() $= (K, \overline{k})$ and Sign$(M, \overline{k}) = \Sigma$ then Σ has the form $\overline{C}\,\overline{S}$ by definition of Sign, so the first step of Verify does not return; $K \in G$, so the second step of Verify does not return; R in the third step of Verify is the same as the R computed in Sign; and $C = H(RM)$ by definition of Sign, so the fourth step of Verify does not return. Hence Verify$(M, \Sigma, K) =$ True.

2.4. Key prefixing. Key prefixing is a generic transformation that converts any signature scheme into what I call a "key-prefixed" signature scheme. In short, a key-prefixed signature scheme inserts the public key in front of the message before signing or verifying it.

Key prefixing was described by Menezes and Smart in [30, proof of Theorem 6] as obviously eliminating "key-substitution attacks". Menezes and Smart argued that preventing these "attacks" is useful, even though these "attacks" do not violate the standard definition of signature security. On the other hand, Menezes and Smart did not argue that key prefixing was useful for Schnorr signatures: [30, Theorem 7] states (under standard hypotheses) that Schnorr signatures *without* key prefixing already resist key-substitution "attacks".

Key prefixing for a Schnorr-like signature system was later recommended in [8], as discussed in Section 1.

2.5. Key-tag prefixing. For Sections 3 and 4 it is useful to consider a more general transformation. This transformation is parametrized by a positive integer t and a deterministic algorithm $\mathrm{Tag} : \{0,1\}^* \to \{0,1\}^t$; what the transformation does is insert $\mathrm{Tag}(K)$ in front of M. Formally, for any signature system X, the transformed signature system $\mathrm{Tag} + X$ is defined as follows:

- The transformed set of secret keys is the set of all concatenations $T\,k$ where $T \in \{0,1\}^t$ and k is in the original set of secret keys.
- The transformed set of public keys is the original set of public keys.
- The transformed set of messages is the set of all strings M such that, for every t-bit string T, the concatenation $T\,M$ is in the original set of messages.
- The transformed set of signatures is the original set of signatures.
- The transformed key-generation algorithm $\mathrm{Gen}_{\mathrm{Tag}}$ calls the original key-generation algorithm Gen, obtaining (K, k), and returns $(K, \mathrm{Tag}(K)\,k)$.
- The transformed signing algorithm $\mathrm{Sign}_{\mathrm{Tag}}$, on input $(M, T\,k)$ where $T \in \{0,1\}^t$, returns the output of $\mathrm{Sign}(T\,M, k)$.
- The transformed verification algorithm $\mathrm{Verify}_{\mathrm{Tag}}$, on input (M, Σ, K), returns the output of $\mathrm{Verify}(\mathrm{Tag}(K)\,M, \Sigma, K)$.

It is easy to check that $\mathrm{Tag} + X$ is a signature system. The output of $\mathrm{Gen}_{\mathrm{Tag}}$ has the form $(K, \mathrm{Tag}(K)\,k)$, and $\mathrm{Sign}_{\mathrm{Tag}}(M, \mathrm{Tag}(K)\,k)$ has the same output Σ as $\mathrm{Sign}(\mathrm{Tag}(K)\,M, k)$. This output satisfies $\mathrm{Verify}(\mathrm{Tag}(K)\,M, \Sigma, K) = \mathrm{True}$ and therefore satisfies $\mathrm{Verify}_{\mathrm{Tag}}(M, \Sigma, K) = \mathrm{True}$.

One special case is that Tag is injective on the set of public keys; this transformation is then, by definition, key prefixing. Another special case that $t = 0$, i.e., $\mathrm{Tag} = \mathrm{Empty}$, where Empty is the unique function $\{0,1\}^* \to \{0,1\}^0$; the key-tag-prefixed signature system $\mathrm{Empty} + X$ with these empty tags is the same as the original signature system X.

When the original signature system is the classic Schnorr signature system, this transformation ends up replacing $H(RM)$ with $H(R\,\mathrm{Tag}(K)\,M)$ in both Sign and Verify, along with including $\mathrm{Tag}(K)$ in the secret key for use by Sign. If very little space is available for the secret key then $\mathrm{Tag}(K)\,k$ can be compressed to simply k, with $\mathrm{Tag}(K) = \mathrm{Tag}(kB)$ recomputed from k on demand. The same type of compression applies more generally to any signature system that allows recomputation of K, or at least $\mathrm{Tag}(K)$, from k. I am not saying that this recomputation is free; the cost of this recomputation was highlighted by Hamburg in [20].

3 Security

This section builds a 1-key attack $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$ against the classic Schnorr signature system $\mathrm{Schnorr}_{G,B,H}$, using an N-key attack A against the key-tag-prefixed system $\mathrm{Tag} + \mathrm{Schnorr}_{G,B,H}$. The cost of $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$

is practically identical to the cost of A, and Theorem 3.6 states that *if the* Tag *function is injective on G* then the success probability of $\text{ReRandomize}_{N,\text{Tag}}(A)$ equals the success probability of A. In other words, key-prefixed Schnorr is at least as secure against multi-key attacks as the classic Schnorr system is against single-key attacks.

The special case $\text{ReRandomize}_{N,\text{Empty}}(A)$ is, aside from irrelevant details, exactly the construction given by GMLS. However, in the same special case, the probability statement is wrong. Section 4 gives an example of an attack A for which $\text{ReRandomize}_{N,\text{Empty}}(A)$ has success probability 0 while A has much larger success probability.

3.1. Single-key security of signature systems. A 1-key attack A against a signature system X receives two inputs: first, a public key K; second, an oracle for $M \mapsto \text{Sign}(M, k)$. Here (K, k) is output by Gen. The attack is **successful** if its output is (M, Σ, K) where

- M is a message,
- Σ is a signature,
- $\text{Verify}(M, \Sigma, K) = \text{True}$, and
- M was not a query to the oracle.

Define $\text{PrForge}_{1,X}(A)$ as the probability that A is successful over all coin flips in Gen, in each call to Sign, and in A itself.

I have deviated slightly from the traditional syntax for attacks: specifically, the attack is required to repeat the same public key K as part of its output. This has the notational advantage that a 1-key attack is, as one would expect, the special case $N = 1$ of an N-key attack, defined below.

3.2. Multi-key security of signature systems. An N-key attack against a signature system X receives $2N$ inputs: public keys $K_1, K_2, \ldots, K_N$ and oracles $O_1, O_2, \ldots, O_N$, where O_i is an oracle for $M \mapsto \text{Sign}(M, k_i)$. Here each (K_i, k_i) is an independent uniform random output from Gen. The attack is **successful** if its output is (M, Σ, K') where

- M is a message,
- Σ is a signature,
- $K' \in \{K_1, K_2, \ldots, K_N\}$,
- $\text{Verify}(M, \Sigma, K') = \text{True}$, and
- M was not a query to O_i for any i with $K_i = K'$.

Define $\text{PrForge}_{N,X}(A)$ as the probability that A is successful over all coin flips in the N calls to Gen, in each call to Sign, and in A itself.

Note that M *is* allowed to have been a query to O_i if $K_i \neq K'$. For example, asking a user to sign a message, and then figuring out a signature of the same message under another user's key, *does* count as a successful attack. However, if the keys happen to collide then this does *not* count as a successful attack. This key-collision convention makes the theory cleaner (otherwise the attack output needs to be tagged with the index of the victim user, and the probability statement for ReRandomize becomes more complicated), and it does not matter for users: key collisions from legitimate runs of Gen are extremely rare for any signature system of interest.

The GMLS security defintions in [16, page 264] do not make clear what happens if keys collide: the definitions refer to, e.g., “the signing oracle corresponding to the key” as if this were necessarily unique. However, outside this rare case, the definitions in [16] do clearly say that M “may have been a query to any of the other signing oracles”.

3.3. The reduction. Let A be an N-key attack against $\text{Tag} + \text{Schnorr}_{G,B,H}$. Define a 1-key attack $\text{ReRandomize}_{N,\text{Tag}}(A)$ against $\text{Schnorr}_{G,B,H}$ as follows.

Generate independent uniform random $r_1, r_2, \ldots, r_N \in \mathbf{Z}/\ell$. Compute $K_1 = K + r_1 B$, $K_2 = K + r_2 B$, and so on through $K_N = K + r_N B$, where K is the input key. Note that $K_1, K_2, \ldots, K_N$ are independent uniform random elements of the group G. Also compute $\text{Tag}(K_1), \text{Tag}(K_2), \ldots, \text{Tag}(K_N)$.

Recall the definition of $\text{Schnorr}_{G,B,H}$: the input oracle O, on input M, generates a uniform random element $R \in G$ and returns $\overline{C}\,\overline{S}$ where $C = H(RM)$ and $SB = R + CK$. For each $i \in \{1, 2, \ldots, N\}$, define oracle O_i as follows:

- Compute $\overline{C}\,\overline{S} = O(\text{Tag}(K_i)\,M)$, where M is the input. (Now $SB = R + CK$ and $C = H(R\ \text{Tag}(K_i)\,M)$.)
- Compute $S' = S + r_i C$. (Now $S'B = SB + Cr_iB = R + CK_i$.)
- Return $\overline{C}\,\overline{S'}$.

This oracle O_i has exactly the same output as a signing oracle for K_i in the key-tag-prefixed system: it generates a uniform random element $R \in G$ and returns $\overline{C}\,\overline{S'}$ where $C = H(R\ \text{Tag}(K_i)\,M)$ and $S'B = R + CK_i$.

Now run A on inputs $K_1, K_2, \ldots, K_N, O_1, O_2, \ldots, O_N$. If the output of A has the form $(M, \overline{C}\,\overline{S}, K')$ for some $(C, S) \in (\mathbf{Z}/\ell)^2$, and there is some $j \in \{1, 2, \ldots, N\}$ such that $K_j = K'$, then compute $S' = S - r_j C$ and output $(\text{Tag}(K_j)\,M, \overline{C}\,\overline{S'}, K)$. Note that all K_j with $K_j = K'$ have the same r_j, so there is no need to specify a choice of j when there are multiple possibilities.

3.4. Time. There is a standard convention of counting the time for the legitimate user algorithms as part of the time of the attack (with the caveat that this disregards, e.g., the real time saved from having many users generate keys in parallel). With this convention, the time for $\text{ReRandomize}_{N,\text{Tag}}(A)$ is practically identical to the time for A, under minor hypotheses regarding the costs of group operations etc.:

- Key generation: A receives N keys, which are generated with N scalar multiplications $k_i B$. $\text{ReRandomize}_{N,\text{Tag}}(A)$ receives one key, which was generated with one scalar multiplication, and then adds N scalar multiples $r_i B$. For all groups G of interest, the cost of addition is negligible compared to the cost of scalar multiplication, so the cost of $N + 1$ scalar multiplications and N additions is practically identical to the cost of N scalar multiplications. Both A and $\text{ReRandomize}_{N,\text{Tag}}(A)$ also involve N calls to the Tag function.

- Signing: Each oracle call from A involves a scalar multiplication to generate R, a hash call with input $R\,\mathrm{Tag}(K_i)\,M$, and some easy arithmetic in $\mathbf{Z}/\ell$. Each oracle call from $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$ involves a scalar multiplication to generate R, a hash call with input $R\ \mathrm{Tag}(K_i)\,M$, and slightly more arithmetic in $\mathbf{Z}/\ell$, since the initially computed S is then adjusted to obtain S'.
- Output: $\mathrm{ReRandomize}_{N,\mathrm{Tag}}$ looks up K in a K_i table and performs some easy final arithmetic in $\mathbf{Z}/\ell$.

3.5. Success probability. The following theorem is the main theorem of this paper. See Section 4 for the importance of the Tag-injectivity hypothesis.

Theorem 3.6. *If* Tag *is injective on G then*

$$\mathrm{PrForge}_{1,\mathrm{Schnorr}_{G,B,H}}(\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)) = \mathrm{PrForge}_{N,\mathrm{Tag}\,+\,\mathrm{Schnorr}_{G,B,H}}(A).$$

Proof. The public keys $K_1, K_2, \dots, K_N$ computed by $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$ are, as noted above, independent uniform random elements of G, distributed identically to the public keys produced by N independent uniform random outputs from Gen; and the oracles $O_1, O_2, \dots, O_N$ are the corresponding signing oracles. The run of A inside $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$ therefore succeeds with probability exactly $\mathrm{PrForge}_{N,\mathrm{Tag}\,+\,\mathrm{Schnorr}_{G,B,H}}(A)$. I will show that $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$ succeeds if and only if A succeeds; consequently $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$ succeeds with probability $\mathrm{PrForge}_{N,\mathrm{Tag}\,+\,\mathrm{Schnorr}_{G,B,H}}(A)$ as claimed.

If $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$ succeeds then it must have produced output of the form $(\mathrm{Tag}(K_j)\,M, \overline{C}\,\overline{S'}, K)$, so A must have produced output $(M, \overline{C}\,\overline{S}, K_j)$ with $S' = S - r_jC$. Also $\mathrm{Verify}(\mathrm{Tag}(K_j)\,M, \overline{C}\,\overline{S'}, K) = \mathrm{True}$, so $C = \underline{H}(\underline{R}\ \mathrm{Tag}(K_j)\,M)$ where $R = S'B - CK = SB - CK_j$; consequently $\mathrm{Verify}_{\mathrm{Tag}}(M, \overline{C}\,\overline{S}, K_j) = \mathrm{True}$. Finally, $\mathrm{Tag}(K_j)\,M$ was not a query to O, so M cannot have been a query to O_i for any i with $K_i = K_j$: a query of M to O_i would have produced a query of $\mathrm{Tag}(K_i)\,M$ to O, i.e., a query of $\mathrm{Tag}(K_j)\,M$ to O. In short, A's output passes verification, and the message was not a query to any signing oracle with the same key.

Conversely, assume that A succeeds. Then A must have produced output of the form (M, Σ, K_j) for some $\underline{j}$, and this output must have passed verification; i.e., Σ must have the form $\overline{C}\,\overline{S}$, where the group element $R = SB - CK_j$ satisfies $C = \underline{H}(R\,\mathrm{Tag}(K_j)\,M)$. $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$ then produces output $(\mathrm{Tag}(K_j)\,M, \overline{C}\,\overline{S'}, K)$ where $S' = S - r_jC$. This output passes verification since $R = S'B - CK$.

All that remains is to show that $\mathrm{Tag}(K_j)\,M$ was not a query to O from $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$. This is where the injectivity of Tag is critical.

The queries from $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$ to O all have the form $\mathrm{Tag}(K_i)\,M'$ where M' is a query from A to O_i; I'm using the notation M' here to avoid confusion with the message M output by A.

Suppose that $\mathrm{Tag}(K_j)\,M$ was a query to O from $\mathrm{ReRandomize}_{N,\mathrm{Tag}}(A)$. Then $\mathrm{Tag}(K_j)\,M = \mathrm{Tag}(K_i)\,M'$ for some query M' from A to some O_i. This implies

$\text{Tag}(K_j) = \text{Tag}(K_i)$ and $M = M'$; Tag is injective on G by hypothesis, so $K_j = K_i$; hence M was a query from A to O_i for some i with $K_j = K_i$. However, the definition of success says that M was not a query from A to O_i for any i with $K_i = K_j$. Contradiction. Hence $\text{Tag}(K_j)M$ was not a query to O from $\text{ReRandomize}_{N,\text{Tag}}(A)$. □

4 The importance of injectivity

This section analyzes the consequences for Theorem 3.6 if Tag is allowed to be non-injective on G.

4.1. Injectivity on the set of generated keys. The critical step in the logic is from $\text{Tag}(K_j) = \text{Tag}(K_i)$ to $K_j = K_i$. This step does not need Tag to be injective on all of G; it still works if Tag is injective on $\{K_1, K_2, \ldots, K_N\}$.

For example, if $N = 2^{60}$ and $t = 160$, then a uniform random function Tag collides on $\{K_1, K_2, \ldots, K_N\}$ with probability at most $2^{60}(2^{60} - 1)/2^{161} < 2^{-41}$, since there are at most 2^{60} distinct elements of $\{K_1, K_2, \ldots, K_N\}$. One can reasonably argue that 2^{-41} is acceptable as an attack probability.

This approach will also work for any specific function Tag where the distribution of $\text{Tag}(K)$ is sufficiently well understood. Specifically, if the elements of $\{0,1\}^t$ are taken on by exactly $D_0, D_1, D_2, \ldots, D_{2^t-1}$ values of K respectively, then two keys collide with probability $\sum_i D_i^2/\ell^2$, so N keys include a collision with probability at most $\binom{N}{2}\sum_i D_i^2/\ell^2$. If $D_i \le D$ for each i then $\sum_i D_i^2 \le D\sum_i D_i = D\ell$ so N keys include a collision with probability at most $\binom{N}{2}D/\ell$.

For example, say G is an encoding of points $(x, y) \in \{0, 1, \ldots, q-1\}^2$ on a complete twisted Edwards curve over a prime field $\mathbf{F}_q$. Define $\text{Tag}(K)$ as y mod 2^t, viewed as a t-bit string, when K is the encoding of (x, y). Any particular value of $\text{Tag}(K)$ is consistent with at most $\lceil q/2^t \rceil$ elements $y \in \{0, 1, \ldots, q-1\}$, and thus at most $2\lceil q/2^t \rceil$ elements $K \in G$. Hence N keys include a collision with probability at most $\binom{N}{2}2\lceil q/2^t \rceil/\ell$. As a specific example, Ed25519 has $q < 2^{255}$ and $\ell > 2^{252}$, so 2^{60} keys include a collision with probability below 2^{-37} for $t = 160$.

4.2. Empty tags. The approach above obviously breaks down for short tag lengths, and in particular for empty tags. The problem goes far beyond one step in the logic: if tags collide then the important conclusion—namely, that $\text{ReRandomize}_{N,\text{Tag}}(A)$ succeeds if A succeeds—is simply wrong. As an extreme example, take empty tags, and consider the following very slow 2-key attack A:

- Ask the first key for a signature of the message "yes".
- Use the baby-step-giant-step algorithm to compute the discrete logarithm of the second key.
- Apply the normal signing procedure to forge a signature of the message "yes" under the second key.

This attack A has success probability almost exactly 1: specifically, A succeeds if and only if the two keys are distinct, and this occurs with probability exactly $1 - 1/\ell$. However, the attack $\text{ReRandomize}_{2,\textbf{Empty}}(A)$ has success probability exactly 0: it outputs a signature of "yes", but that's not a forgery, since earlier it asked for a signature of "yes".

4.3. Other reductions. Could ReRandomize be modified somehow to work for empty tags? The obstacles are formidable:

- The signature system hashes messages, and I can't imagine how the reduction could modify hash inputs, so the reduction will have to produce a forgery on the same message "yes" forged by A.
- For this to be a forgery, the reduction has to retroactively avoid asking for a signature on "yes". How can the reduction predict that "yes" will be the message to be forged?
- Even if the reduction does somehow predict this, how will it answer A's signing query for "yes" from the first user? The reduction can't simply skip that query—maybe A actually does some interesting computation on the signature result, influencing its final output.

The original GMLS paper doesn't try to address any of these obstacles. It writes down essentially $\text{ReRandomize}_{N,\textbf{Empty}}(A)$, and correctly observes that success in A implies that the output of $\text{ReRandomize}_{N,\textbf{Empty}}(A)$ is a valid signature. It then leaps to the conclusion that the output is a "valid forgery", without checking whether the message was signed before.

4.4. Possible paths forward. To summarize, it is unclear whether the classic Schnorr signature system is as secure for multiple keys as for a single key. Here are several possible ways to gain clarity:

- Somehow find a way around the above obstacles and build a security proof. I don't expect this to happen.
- Formalize the obstacles by constructing an artificial (G, H) for which the system is less secure for multiple keys than for single keys. Unfortunately, the value of such a formalization is quite unclear: it leaves open the possibility of a security proof for real-world choices of (G, H). For comparison, Shoup in [**33**, Section 5] presented a similar formalization of an obstacle to a generic security proof for OAEP, but in [**33**, Section 7.2] presented a security proof for the specific OAEP construction of maximum interest, namely RSA-3-OAEP.
- Find an actual multi-key attack against the system with real-world choices of (G, H).
- After serious cryptanalytic attention to possible multi-key attacks against the system, conclude that the system is in fact secure for multiple keys, despite the lack of a security proof.

The only possibilities that would build confidence in the security of the system are the first and last possibilities, both of which seem quite far from the current situation. Clearly the safest recommendation is to switch to the key-prefixed variant of the system.

5 Generalizations

In previous sections I focused on the classic Schnorr signature system and its key-tag-prefixed variant. However, there are several reasons that modern signature systems have deviated in further ways from Schnorr's system; see generally [7] for background. This section briefly analyzes the multi-key security of a broader class of signature systems.

5.1. Derandomization. Instead of generating r as a uniform random element of $\mathbf{Z}/\ell$, a *derandomized* signer generates r as $F(z, M)$, where z is another secret key.

If the random function $M \mapsto F(z, M)$ is indistinguishable from a uniform random function $\{0,1\}^* \to \mathbf{Z}/\ell$, a standard "PRF" design goal, then the derandomized signer is indistinguishable from a signer that generates a uniform random r for each M, while remembering which r was used for each M. Carrying out an attack against a derandomized signer is thus indistinguishable from carrying out a restricted attack against a classic signer, where the restriction is that the attack does not ask for multiple signatures on the same message.

In the multi-key setting, consider N signers with secrets $z_1, z_2, \ldots, z_N$, where the ith signer generates r as $F(z_i, M)$. Indistinguishability of these signers from classic signers, with the same restriction of asking each signer for at most one signature on each M, follows from indistinguishability of the function $i, M \mapsto F(z_i, M)$ from a uniform random function. The latter indistinguishability is an N-key version of the standard PRF goal; it cannot be broken with probability more than N times larger than the probability of breaking the standard PRF goal. There are many standard ways to design a PRF for a very high security level using a large key z, absorbing the impact of this factor of N (for any plausible size of N) and preserving the overall security of the signature system.

5.2. Key derivation. Ed25519 [8] and the more general EdDSA [10] actually generate (k, z) by hashing a single master secret. The indistinguishability of these outputs from independent uniform random secrets k and z is another standard pseudorandomness assumption on the hash function. As above, there is at most a loss factor of N in moving to N keys.

5.3. Limited key ranges. EdDSA has a parameter "$n - c$" specifying the number of bits used for the secret scalar k. Obviously there are attacks if this number of bits is chosen too small: specifically, [10] says that "kangaroo" attacks find k using approximately $1.36\sqrt{2^{n-c}}$ group additions.

If A is a 1-key attack that succeeds with probability p against keys chosen uniformly from a restricted interval, say $\{0, 1, \ldots, \delta - 1\}$ for some $\delta < \ell$, then it succeeds with probability at least $(\delta/\ell)p$ against keys chosen uniformly from the entire set $\mathbf{Z}/\ell$. This is acceptably tight for, e.g., $\delta/\ell \approx 1/2$ or $\delta/\ell \approx 1/4$.

However, a restricted interval has more serious consequences for the multi-key security proof in this paper. $\text{ReRandomize}_{N,\text{Tag}}$ produces N keys in the interval $\{0, 1, \ldots, \delta - 1\}$ with probability just $(\delta/\ell)^N$; this is useless for large N as soon as δ is noticeably smaller than ℓ.

A tighter reduction starts from a uniform random k in $\{0, \ldots, \beta - 1\}$, and then rerandomizes k as $(k + r_1, k + r_2, \ldots, k + r_N)$ for independent uniform random $r_1, r_2, \ldots, r_N$ in $\{-\beta + 1, -\beta + 2, \ldots, \delta - 1\}$. This produces any particular vector $(k_1, k_2, \ldots, k_N) \in \{0, 1, \ldots, \delta - 1\}^N$ with probability $1/(\delta + \beta - 1)^N$, i.e., more than $\exp(-1)$ times uniform if one chooses $\beta = \lceil \delta/N \rceil$. Consequently the N-key security of an interval of length δ is more than $\exp(-1)$ times the 1-key security of an interval of length $\lceil \delta/N \rceil$.

This is still not tight. Could there be, e.g., an algorithm that finds 1 of N discrete logarithms, each known to be in a particular interval of length $\approx 0.5\ell$, in time $O(\sqrt{\ell/N})$?

To avoid having to think about this question, I suggest that signers take enough bits to guarantee that the distribution of k is indistinguishable from the uniform distribution on $\mathbf{Z}/\ell$. For example, in Ed25519, ℓ is approximately $2^{252} + 2^{124.4}$, and taking $n - c = 252$ (specifically $n = 255$ and $c = 3$) is tantamount to taking $\delta/\ell \approx 1 - 2^{-127.6}$, so $(\delta/\ell)^N > 1 - 2^{-50}$ even for N as large as 2^{75}.

5.4. More encodings of (R, C, S). In Schnorr's system, the per-signature group element R, hash $C = H(RM)$, and integer $S = r + Ck$ are transmitted as $\overline{C}\,\overline{S}$; the verifier computes R as $SB - CK$ and then checks that $C = H(RM)$.

An alternative is to transmit $R\overline{S}$. The verifier computes $C = H(RM)$ and checks that $SB = R + CK$. This alternative is used in, e.g., Ed25519 and EdDSA.

In both cases, the reconstruction of (R, C, S) is public. After verifying a signature in the form $\overline{C}\,\overline{S}$ one also knows a signature in the form $R\overline{S}$, and vice versa. An algorithm to attack signatures in one form, for any number of keys, is thus easily transformed into an equally effective algorithm to attack signatures in the other form at practically identical speed.

5.5. More general linear equations. In Schnorr's system, S is a linear combination of the per-signature secret r and the long-term secret k, with coefficients 1 and $H(RM)$. More generally, one can replace the coefficients by any functions of R and M, say $H_0(RM)$ and $H_1(RM)$. Given a signature $R\overline{S}$, the verifier checks that $SB = H_0(RM)R + H_1(RM)K$.

This is what GMLS call "Schnorr-like signatures". The ReRandomize construction is easily adapted to this level of generality. However, at least from a security perspective, this generalization is content-free. One can easily encode S as the ratio $S' = S/H_0(RM)$ in $\mathbf{Z}/\ell$; for any reasonable choice of H_0 one does not have to worry about the possibility of $H_0(RM)$ being zero. The verifier, given R and S', now checks $S'B = R + H(RM)K$, where by definition $H(RM) = H_1(RM)/H_0(RM)$. In other words, this encoding of "Schnorr-like" signatures is identical to classic Schnorr signatures for this function H.

Struik in [35] cited GMLS for "Schnorr and ECDSA type schemes", but GMLS say that "this definition of Schnorr-like signatures excludes the popular signature schemes DSA and ECDSA". Recall that the ECDSA verification equation is $(H(M)/S)B + (x(R)/S)K = R$, for a function x whose details are not relevant here; equivalently, $H(M)B + x(R)K = SR$; equivalently, $B = S'R + H'(RM)K$ where $S' = S/H(M)$ and $H'(RM) = -x(R)/H(M)$. This is an example of Schnorr *except* that S' is in front of R rather than B.

This change of position is important: there is no obvious way to adjust S for a long-term rerandomization of K.

References

[1] Gilles Barthe, Edvard Fagerholm, Dario Fiore, John Mitchell, Andre Scedrov, Benedikt Schmidt, *Automated analysis of cryptographic assumptions in generic group models*, in Crypto 2014 [**18**] (2014), 95–112. URL: https://eprint.iacr.org/2014/458. Citations in this document: §1.4.

[2] Mihir Bellare (editor), *Advances in cryptology—CRYPTO 2000, proceedings of the 20th annual international cryptology conference held in Santa Barbara, CA, August 20–24, 2000*, Lecture Notes in Computer Science, 1880, Springer, 2000. ISBN 3-540-67907-3. MR 2002c:94002. See [12].

[3] Mihir Bellare, Phillip Rogaway, *Optimal asymmetric encryption—how to encrypt with RSA*, in Eurocrypt 1994 [**14**] (1995), 92–111. URL: http://cseweb.ucsd.edu/~mihir/papers/oaep.html. Citations in this document: §5.

[4] Daniel J. Bernstein, *Understanding brute force*, ECRYPT STVL Workshop on Symmetric Key Encryption (2005). URL: http://cr.yp.to/papers.html#bruteforce. Citations in this document: §1.

[5] Daniel J. Bernstein, *Some thoughts on security after ten years of qmail 1.0*, in Proceedings of Computer Security Architecture Workshop (CSAW) 2007 (2007). URL: http://cr.yp.to/papers.html#qmailsec. Citations in this document: §1.4.

[6] Daniel J. Bernstein, *The fundamental goal of "provable security"* (2013). URL: http://cr.yp.to/talks/2013.01.23/slides.pdf. Citations in this document: §1.4.

[7] Daniel J. Bernstein, *How to design an elliptic-curve signature system* (2014). URL: http://blog.cr.yp.to/20140323-ecdsa.html. Citations in this document: §5.

[8] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, Bo-Yin Yang, *High-speed high-security signatures*, in CHES 2011 [**32**] (2011); see also newer version [**9**]. URL: http://ed25519.cr.yp.to/ed25519-20110705.pdf. Citations in this document: §2.4, §5.2.

[9] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, Bo-Yin Yang, *High-speed high-security signatures*, Journal of Cryptographic Engineering **2** (2012), 77–89; see also older version [**8**]. URL: http://ed25519.cr.yp.to/ed25519-20110926.pdf. Citations in this document: §1.2.

[10] Daniel J. Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe, Bo-Yin Yang, *EdDSA for more curves* (2015). URL: https://eprint.iacr.org/2015/677. Citations in this document: §5.2, §5.3.

[11] Daniel J. Bernstein, Tanja Lange, *Computing small discrete logarithms faster*, in Indocrypt 2012 [**17**] (2012), 317–338. URL: https://eprint.iacr.org/2012/458. Citations in this document: §3.

[12] Ingrid Biehl, Bernd Meyer, Volker Müller, *Differential fault attacks on elliptic curve cryptosystems (extended abstract)*, in Crypto 2000 [**2**] (2000), 131–146. URL: http://lecturer.ukdw.ac.id/vmueller/publications.php. Citations in this document: §2.2.

[13] Sanjit Chatterjee, Alfred Menezes, Palash Sarkar, *Another look at tightness*, in SAC 2011 [**31**] (2012), 293–319. URL: https://eprint.iacr.org/2011/442. Citations in this document: §1.

[14] Alfredo De Santis (editor), *Advances in cryptology—EUROCRYPT '94, workshop on the theory and application of cryptographic techniques, Perugia, Italy, May 9–12, 1994, proceedings*, Lecture Notes in Computer Science, 950, Springer, 1995. ISBN 3-540-60176-7. MR 98h:94001. See [3].

[15] Adrian E. Escott, John C. Sager, Alexander P. L. Selkirk, Dimitrios Tsapakidis, *Attacking elliptic curve cryptosystems using the parallel Pollard rho method*, CryptoBytes **4** (1999). URL: ftp://ftp.rsa.com/pub/cryptobytes/crypto4n2.pdf. Citations in this document: §3.

[16] Steven D. Galbraith, John Malone-Lee, Nigel P. Smart, *Public-key signatures in the multi-user setting*, Information Processing Letters **83** (2002), 263–266. Citations in this document: §1, §1, §1.1, §3.2, §3.2.

[17] Steven Galbraith, Mridul Nandi (editors), *Progress in cryptology—Indocrypt 2012: 13th international conference on cryptology in India, Kolkata, India, December 9–12, 2012, proceedings*, Lecture Notes in Computer Science, 7668, Springer, 2012. See [11].

[18] Juan A. Garay, Rosario Gennaro (editors), *Advances in cryptology—CRYPTO 2014, 34th annual cryptology conference, Santa Barbara, CA, USA, August 17–21, 2014, proceedings, part I*, Lecture Notes in Computer Science, 8616, Springer, 2014. ISBN 978-3-662-44370-5. See [1].

[19] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM Journal on Computing **17** (1988), 281–308. ISSN 0097-5397. MR 89e:94009. URL: http://theory.lcs.mit.edu/~rivest/publications.html. Citations in this document: §2.1.

[20] Mike Hamburg, *Re: [Cfrg] EC signature: next steps* (2015). URL: https://mailarchive.ietf.org/arch/msg/cfrg/af170b60rLyNZUHBMOPWxcDrVRI. Citations in this document: §1.2, §2.5.

[21] Yvonne Hitchcock, Paul Montague, Gary Carter, Ed Dawson, *The efficiency of solving multiple discrete logarithm problems and the implications for the security of fixed elliptic curves*, International Journal of Information Security **3** (2004), 86–98. Citations in this document: §3.

[22] Tibor Jager, Jörg Schwenk, Juraj Somorovsky, *Practical invalid curve attacks on TLS-ECDH*, in ESORICS 2015 (2015). URL: https://www.nds.rub.de/research/publications/ESORICS15/. Citations in this document: §2.2.

[23] Joe Kilian (editor), *Advances in cryptology—CRYPTO 2001, 21st annual international cryptology conference, Santa Barbara, California, USA, August 19–23, 2001, proceedings*, Lecture Notes in Computer Science, 2139, Springer, 2001. ISBN 3-540-42456-3. MR 2003d:94002. See [33].

[24] Neal Koblitz, *Another look at automated theorem-proving*, Journal of Mathematical Cryptology **1** (2007), 385–403. URL: https://eprint.iacr.org/2007/401. Citations in this document: §1.4.

[25] Neal Koblitz, *Another look at automated theorem-proving. II*, Journal of Mathematical Cryptology **5** (2011), 205–224. URL: https://eprint.iacr.org/2011/485. Citations in this document: §1.4.

[26] Neal Koblitz, Alfred Menezes (editors), *Another look at provable security*, web site, accessed 9 October 2015. URL: http://anotherlook.ca/. Citations in this document: §5.

[27] Hugo Krawczyk, *HMQV: a high-performance secure Diffie-Hellman protocol*, in Crypto 2005 [**34**] (2005), 546–566. URL: https://eprint.iacr.org/2005/176. Citations in this document: §1.2.

[28] Fabian Kuhn, Rene Struik, *Random walks revisited: extensions of Pollard's rho algorithm for computing multiple discrete logarithms*, in SAC 2001 [**36**] (2001), 212–229. URL: http://www.distcomp.ethz.ch/publications.html. Citations in this document: §3.

[29] Alfred Menezes, *Another look at HMQV*, Journal of Mathematical Cryptology **1** (2007), 47–64. URL: https://eprint.iacr.org/2005/205. Citations in this document: §2.2.

[30] Alfred Menezes, Nigel P. Smart, *Security of signature schemes in a multi-user setting*, Designs, Codes and Cryptography **33** (2002), 261–274. Citations in this document: §2.4, §2.4.

[31] Ali Miri, Serge Vaudenay (editors), *Selected areas in cryptography: 18th international workshop, SAC 2011, Toronto, Canada, August 11–12, 2011, revised selected papers*, Lecture Notes in Computer Science, 7118, Springer, 2012. See [13].

[32] Bart Preneel, Tsuyoshi Takagi (editors), *Cryptographic hardware and embedded systems—CHES 2011, 13th international workshop, Nara, Japan, September 28–October 1, 2011, proceedings*, Lecture Notes in Computer Science, Springer, 2011. ISBN 978-3-642-23950-2. See [8].

[33] Victor Shoup, *OAEP reconsidered*, in Crypto 2001 [**23**] (2001), 239–259. URL: http://shoup.net/papers/oaep.pdf. Citations in this document: §5, §4.4, §4.4.

[34] Victor Shoup (editor), *Advances in cryptology—CRYPTO 2005, 25th annual international cryptology conference, Santa Barbara, California, USA, August 14–18, 2005, proceedings*, Lecture Notes in Computer Science, 3621, Springer, 2005. ISBN 3-540-28114-2. See [27].

[35] Rene Struik, *Re: [Cfrg] EC signature: next steps* (2015). URL: https://mailarchive.ietf.org/arch/msg/cfrg/TOWH1DSzB-PfDGK8qEXtF3iC6Vc. Citations in this document: §1, §1.2, §5.5.

[36] Serge Vaudenay, Amr M. Youssef (editors), *Selected areas in cryptography: 8th annual international workshop, SAC 2001, Toronto, Ontario, Canada, August 16–17, 2001, revised papers*, Lecture Notes in Computer Science, 2259, Springer, 2001. ISBN 3-540-43066-0. MR 2004k:94066. See [28].