Generic Groups, Collision Resistance, and ECDSA

Daniel R. L. Brown*

Certicom Research, Canada
dbrown@certicom.com
February 26, 2002

Abstract. Proved here is the sufficiency of certain conditions to ensure the Elliptic Curve Digital Signature Algorithm (ECDSA) existentially unforgeable by adaptive chosen-message attacks. The sufficient conditions include (i) a uniformity property and collision-resistance for the underlying hash function, (ii) pseudo-randomness in the private key space for the ephemeral private key generator, (iii) generic treatment of the underlying group, and (iv) a further condition on how the ephemeral public keys are mapped into the private key space. For completeness, a brief survey of necessary security conditions is also given. Some of the necessary conditions are weaker than the corresponding sufficient conditions used in the security proofs here, but others are identical. Despite the similarity between DSA and ECDSA, the main result is not appropriate for DSA, because the fourth condition above seems to fail for DSA. (The corresponding necessary condition is plausible for DSA, but is not proved here nor is the security of DSA proved assuming this weaker condition.) Brickell et al. [11], Jakobsson et al. [29] and Pointcheval et al. [44] only consider signature schemes that include the ephemeral public key in the hash input, which ECDSA does not do, and moreover, assume a condition on the hash function stronger than the first condition above. This work seems to be the first advance in the provable security of ECDSA.

1 Introduction

Koblitz [32] and Miller [40] independently proposed elliptic curve cryptography in 1985. The National Institute of Standards and Technology (NIST) proposed the Digital Signature Algorithm (DSA) in 1991 (see [10]). Vanstone [53] proposed an elliptic curve analogue of DSA, the Elliptic Curve Digital Signature Algorithm (ECDSA), in 1992. The current form of ECDSA was proposed by the IEEE P1363 working group in 1995. Koblitz describes ECDSA in his book [33], as do Blake, Seroussi and Smart in their book [6]. Johnson and Menezes [30] give an excellent survey on ECDSA. Detailed specifications of ECDSA are given in several approved standards: ISO 14888-3 [28], IEEE Std 1363-2000 [27], ANSI X9.62 [2], and FIPS 186-2 [21] (see also [47]). Applications of ECDSA are proposed in several other standards such as WAP WTLS, IETF S/MIME, IETF TLS, and IETF IPSEC IKE.

A proof of conditional security ensures that a cryptographic technique meets its security objectives provided that proof's conditions are met. For a security proof to offer real assurance, the security objectives should have some real value. Furtheremore, the milder a security proof's conditions, the greater its security assurance. Thus strong objectives and mild conditions are desirable. Provable security should not be regarded as a substitute for conventional security (failure of substantial cryptanalytic effort), but rather as a supplement to it. ECDSA's conventional security is fairly well-established. In this paper, a degree of provable security for ECDSA is established for the first time.

Until now, there has been little work on provable security applicable to DSA or ECDSA even though they have both withstood concerted cryptanalytic effort fairly well. Although Pointcheval and Stern [44] give security proofs for ElGamal-like digital signature schemes very similar to ECDSA and DSA, their proof technique does not seem to apply to ECDSA and DSA *per se*. The main obstacle seems to be that ECDSA and DSA do not include the ephemeral public key[1] in the input to the hash function[2]. We overcome this obstacle by using different conditions that allow proof techniques for which inclusion of the ephemeral public key is unneeded. Our conditions on the hash function are *weaker* than [44]: we do not model the hash function by a random oracle; instead we assume collision resistance, a certain uniformity property and an additional minor property. Our condition for the group is *stronger* than [44]: we use more than just the intractability of the elliptic curve discrete logarithm; instead we model the group [48] generically. On balance, our conditions are roughly equal[3] to [44], since one condition is weaker and the other

* Supported in part by a National Science and Engineering Research Council of Canada Industrial Research Fellowship

[1] The ephemeral public key is generated for use with at most one message.

[2] Schnorr signatures [39], for example, include the ephemeral public key in the input to the hash function.

[3] More strictly speaking, the conditions are incomparable.

is stronger. Jakobsson and Schnorr [29] give security proofs using techniques that might extend to ECDSA upon further consideration, but they use a stronger set of the conditions[4]. Of course, digital signature schemes of many other kinds have security proofs [4, 5, 12, 15–17, 20, 24, 43] under a wide variety of conditions.

The remaining sections are organized as follows. Section 2 describes the groups that are used in ECDSA and DSA, defines the term *conversion function* together with some security definitions, and discusses a variant of the generic group model. Section 3 recalls the needed definitions for the security of hash functions. Section 4 briefly recalls the notion of pseudorandom private key generation. Section 5 recalls the definitions for signature schemes, ECDSA, and DSA. Section 6 surveys some conditions that are needed for the security of ECDSA by exhibiting attacks when the conditions fail. Section 7 describes a varian of the generic group model and gives an essential lemma. Finally, Section 8 gives security proofs. The results are summarized in Section 9.

2 Secure Groups

A *secure group* is defined by having an intractable discrete logarithm problem. Furthermore, because ECDSA and DSA use a *conversion function* (defined below), which is closely related to the encoding of the secure group, some additional security conditions are important, namely conditions regarding this conversion function.

The Nechaev-Shoup [41, 48] generic model of a group is an even stronger condition than the intractability of the discrete logarithm problem. Indeed, strictly speaking this condition is unattainable for an efficient group. Nevertheless, assuming such a strong condition is useful for provable security analysis, because it allows proofs using fairly mild conditions on the other features, especially the hash function.

2.1 Discrete Logarithms

Let n be a prime number. In the cryptographic applications we are considering, n will be a security parameter. The larger n, the greater the security. Typically, we work with n such that $n \approx 2^m$ where $m \approx 160, 192, 256, 384$ or 512. The strength of the security offered by such n is said to $m/2$ bits, for reasons that we shall see below.

The group $\mathbb{Z}_n = \mathbb{Z}_n^+ = \{0, 1, \ldots, n-1\}$ of integers modulo n under addition is a cyclic, simple group generated by the element 1. Let $\mathbb{Z}_n^* = \mathbb{Z}_n \setminus \{0\} = \{1, \ldots, n-1\}$ be the set of nonzero elements of $\mathbb{Z}_n$. Let $x \in \mathbb{Z}_n^*$. The function $e_x : \mathbb{Z}_n \to \mathbb{Z}_n : t \mapsto tx$ in $\mathbb{Z}_n$ is *scalar multiplication* to the base x. Scalar multiplication is clearly efficient. The function $l_x : \mathbb{Z}_n \to \mathbb{Z}_n : y \mapsto (yx^{-1} \bmod n)$ is an efficient inverse function of scalar multiplication because $l_x \circ e_x : \mathbb{Z}_n \to \mathbb{Z}_n : t \mapsto t$ and the Euclidean algorithm allows $(x^{-1} \bmod n)$ to be computed efficiently.

For more general groups, the group operation is conventionally referred to as multiplication. Under this convention, the scalar multiplication e_x is an example of *exponentiation* to the base x. The function l_x is the *discrete logarithm* to the base x. In this terminology we say that the discrete logarithm function for the group $\mathbb{Z}_n$ is efficient. Of course, the discrete logarithm function is not efficient for all groups isomorphic to $\mathbb{Z}_n$. Indeed, the difficulty of efficiently computing the discrete logarithm function in certain groups isomorphic to $\mathbb{Z}_n$ is essential ingredient to the security to several cryptographic protocols, such as DSA and ECDSA.

Let $\mathbb{A}_n$ be a group isomorphic to the group $\mathbb{Z}_n$. We shall mostly use additive notation for the group operation of $\mathbb{A}_n$. We shall assume that addition in $\mathbb{A}_n$ is an efficient operation. Let $\mathbb{A}_n^* = \mathbb{A}_n \setminus \{\mathcal{O}\}$, where $\mathcal{O}$ is the zero-element of $\mathbb{A}_n$. Fix some $G \in \mathbb{A}_n^*$. Then $\mathbb{A}_n = \langle G \rangle$ is generated by G. The function $e_G : \mathbb{Z}_n \to \mathbb{A}_n : t \mapsto tG$ is scalar multiplication to the base G. Scalar multiplication is an efficient operation because addition is efficient and the double-and-add algorithm requires roughly $3m/2$ additions on average. The discrete logarithm function $l_G : \mathbb{A}_n \to \mathbb{Z}_n$ to the base G is the inverse of the scalar multiplication function $e_G : \mathbb{Z}_n \to \mathbb{A}_n$.

In certain groups $\mathbb{A}_n$ such as certain elliptic curve groups and multiplicative groups of certain finite fields, it seems infeasible to compute the discrete logarithm function l_G efficiently. The most efficient known algorithms for l_G in these groups $\mathbb{A}_n$ are the Pollard-ρ-algorithm and algorithms with similar running times: about $2^{m/2}$ group operations. We call such a group $\mathbb{A}_n$ a *secure group* of strength $m/2$ bits. (As 2^{80} group operations is considered an infeasible amount of computation with today's resources [14], the security threshold $m \geq 160$ is often recommended.)

In ECDSA and DSA, a pair (A, z) where $A = e_G(z)$ is used as a public key pair. The element A is the public key and the integer z is the private key. More generally, in a pair (A, z) where $A = e_G(z)$, we say that A is the *public value* and z is the *private value*, and these values *correspond* to each other. Given the public value of a pair is it

[4] Namely, they use a strong conditions for both the group and the hash function.

infeasible to determine the corresponding private value, but given the private value of a pair it is easy to determine the public value. Thus e_G is a *one-way permuation* or *isomorphism*. For convenience, we sometimes call the group $\mathbb{A}_n$ the *public (key) group* and the group $\mathbb{Z}_n$ the *private (key) group*. (Occasionally, the private key group has also been called the *log space*, or the *exponent group*.)

The private key group $\mathbb{Z}_n$ also has a ring structure that is used in the ECDSA and DSA signing equations. But the public key group $\mathbb{A}_n$ does not necessarily have such extra structure.

Often $\mathbb{A}_n$ is specified to be subgroup of a larger group. We call this larger group the *supergroup*. The index of the secure group in the supergroup is called the *cofactor*.

It is not the aim of this paper to study the security of any given group $\mathbb{A}_n$. Instead we study how the security of the group $\mathbb{A}_n$ relates to the security of signatures schemes like ECDSA and DSA that use the group. Generally, the study of the security of typical groups $\mathbb{A}_n$ involves a lot number theory and algebra. Our study, however, will involve *reductions* from one type of algorithm to another.

More specifically, we will study the security of ECDSA and DSA under the assumption that $\mathbb{A}_n$ has the best possible security in the following sense. In other words, we will study how ECDSA and DSA can be affected by forging algorithms that behave *generically* with respect to the secure group, meaning that the forging algorithms only invoke the group operations through an oracle. In this setting, we will try to isolate how the security of ECDSA and DSA depends on the security of components other than the group, namely the hash function, the random number generator, and the *conversion function*.

2.2 Conversion Functions

ECDSA and DSA use a *conversion function* $f : \mathbb{A}_n \to \mathbb{Z}_n$. The conversion function must be efficient. The design of ECDSA and DSA is such that the following properties of the conversion function are related to the security. The properties are *almost-bijectivity* and *almost-invertibility*, defined below.

Almost-Bijectivity A conversion function f is *ϵ-clustered* if there exists an element $z_0 \in \mathbb{Z}_n$ such that for $A \in_R \mathbb{A}_n$ the probability that $f(A) = z_0$ is at least ϵ. (Throughout this paper, $\in_R$ means uniformly random selection of an element from a set.) A bijection f is exactly $1/n$-clustered. A constant is 1-clustered. (Heuristic arguments along the lines of [23] suggest that the a random function f is $\log(n)/n$-clustered on average.) A conversion function f is not a bijection if and only if it is ϵ-clustered for some $\epsilon > 1/n$.

A conversion function is *almost-bijective* of strength at least $\log_2(1/\epsilon)$ bits, if it is not ϵ-clustered. Thus, a bijection has strength $\log_2(n)$ bits and a constant has strength 0 bits.

Almost-Invertibility A conversion function f is *almost-invertible* if an *almost inverse* of f is known. An almost inverse `G` is a probabilistic algorithm that behaves like a two-way inverse function to the conversion function f in the following sense. First note that if f had an inverse function $g : \mathbb{Z}_n \to \mathbb{A}_n$, then (i) $f \circ g$ is the identity function on $\mathbb{Z}_n$ and (ii) g applied to random-looking input in $\mathbb{Z}_n$ gives random-looking output in $\mathbb{A}_n$. We expect the almost inverse `G` to exhibit two similar properties. An *almost-inverse* of f is an efficient[5] probabilistic algorithm `G` that takes input $z \in_R \mathbb{Z}_n$ and generates output $A \in \mathbb{A}_n \cup \{\texttt{Invalid}\}$ such that:

1. $A \neq \texttt{Invalid}$ with probability at least[6] $\frac{1}{10}$ as assessed over random choices made for z and `G`.
2. If $A \neq \texttt{Invalid}$ then $f(A) = z$.
3. If independently random inputs $z \in_R \mathbb{Z}_n$ are repeatedly given to `G` until the output $A \neq \texttt{Invalid}$, the resulting probability distribution of such A is indistinguishable[7] from the distribution of random $A \in_R \mathbb{A}_n$.

In other words, `G` will successfully find an preimage of its input at least one time in ten, and furthermore, the resulting preimage exhibits no detectable bias.

[5] With running-time roughly equal to at most 1 (the running-time of computing a group operation in $\mathbb{A}_n$).

[6] The bound 1/10 is quite aribitrary, but should be small. We chose it to be just small enough to be attainable for the ECDSA conversion functions for recommended curves (see §2.3). For some prime order elliptic curves the bound can be increased to $1/(2 + \epsilon)$, slightly improving the bounds that we can prove in §8.

[7] Almost-invertibility could be quantified by the running-time and success probability of a distinguishing adversary. However, for our purposes, it will suffice to insist that distinguishing be infeasible.

Comparison of Almost-Bijectivity and Almost-Invertibility If f is ϵ-clustered for $\epsilon \gg \frac{10}{n}$, then f cannot be almost-invertible for the following reason. Let $z_0 \in \mathbb{Z}_n$ be the element such that $f(A) = z_0$ with probability at least ϵ, which must exist by the ϵ-clustering. We consider two cases which must be distinguished. First, suppose that A is the output of `G` after the above experiment of giving random inputs z until the output $A \neq$ `Invalid`. Thus $f(A) = z$ where z is the last input given. Since z was selected at random, the probablity that $z = z_0$ would normally be $1/n$, but z was selected at random up to the condition that the output was not `Invalid`. Since the probablity that the output is not `Invalid` is at most $1/10$, the probability that $z = z_0$ can increase to at most $10/n$. Second, suppose that A is randomly selected from $\mathbb{A}_n$. The probability that $f(A) = z_0$ is at least ϵ by definition of f being ϵ-clustered. The difference in probabilities that $f(A) = z_0$ in these two experiments allows us to distinguish them. Given the output A of the experiment, compute $f(A)$. If $f(A) = z_0$, then we guess that A is random. If $z_0 \neq f(A)$, then we guess that A is the output of `G`. Since $\epsilon \gg 10/n$, this gives a good distinguisher. Therefore, almost-invertibility implies almost-bijectivity. (Almost-bijectivity almost certainly does not imply almost-invertibility, with the DSA conversion being a likely counterexample, see §2.4.)

2.3 The ECDSA secure groups

In ECDSA the secure group $\mathbb{A}_n$ is a group of order n generated by a point G on an elliptic curve over a finite field of order q where $q \approx tn$ for $t \in \{1, 2, 4\}$. The ECDSA supergroup is the elliptic curve group defined over $\mathbb{F}_q$ and it has order tn, where t is the cofactor. The elliptic curve group usually consists of the set $\{(x, y) : x, y \in \mathbb{F}_q, y^2 = x^3 + ax + b\}$ if q is an odd prime or the set $\{(x, y) : x, y \in \mathbb{F}_q, y^2 + xy = x^3 + ax^2 + b\}$ if q is a power of two. The group operations involve several field operations, see [6, 31, 33, 38, 40, 47] for example. Specific choices for the parameters q, a, b, n, t and G are recommended in [21, 47]. These recommendations help achieve efficiency, security and interoperability.

The ECDSA Conversion Function The conversion function f takes the bit string representation of the x-coordinate of an elliptic curve point, converts it to an integer, and then reduces the integer modulo n. The ECDSA conversion function is almost-invertible, with the following almost-inverse `G`. Given $z \in \mathbb{Z}_n$, choose a random integer i such that (i) $i = z \mod n$ and (ii) the bit representation of i is also the bit representation of a finite field element x. By Hasse's theorem, roughly $1/2$ the field elements x can be x-coordinates of an elliptic curve point. Given such an x-coordinate, solving a quadratic equation gives a y-coordinate of a point $R = (x, y)$ on the curve. For recommended ECDSA curves, at least $1/4$ of all the points belong to the subgroup $\mathbb{A}_n$. Clearly $f(R) = z$. Thus `G` is efficient and is successful with probability at least $1/8$ roughly. Also this `G`, when successful, can obtain any element in $\mathbb{A}_n$ with essentially no bias. Since `G` is an almost-inverse, f is almost-invertible.

2.4 The DSA secure group

In DSA, the supergroup $\mathbb{Z}_p^*$ is the multiplicative group of integers modulo p, where p is some prime such that $n \mid (p-1)$. Usually, $m = 160$, and $p \approx 2^M$ where $M \in \{512, 768, 1024, 2048\}$. Since $n|(p-1) = |\mathbb{Z}_p^*|$, there is an element $G \in \mathbb{Z}_p^*$ of order n. In DSA, the secure group is $\mathbb{A}_n = \langle G \rangle$, with the group operation being integer multiplication modulo p. The cofactor is $(p-1)/n$. Normally, the group operation for the DSA secure group would be written multiplicatively, but our analyses will be unaffected when we write it additively, provided the meaning is unchanged. No confusion will result because our analysis will not focus on the DSA group specifically.

The DSA Conversion Function The DSA conversion function is $f : \langle G \rangle \to \mathbb{Z}_n : a \mapsto (a \bmod n)$. Nguyen and Sharplinksi [42] give partial results showing towards showing the DSA conversion function is almost-bijective. Experiments with small values of n show that the DSA conversion function is as almost-bijective of the same strength as an average random function—we therefore conjecture that this is true. We also conjecture that the DSA conversion function is is *one-way*, and therefore not almost-invertible.

The proofs of Theorem 2 and 4 make essential use the almost-invertibility of the conversion function. Therefore, it is not reasonable to apply them to DSA. Nevertheless, we do not suggest that the one-wayness of the conversion function in DSA is harmful for security. Indeed, one would expect that the one-wayness of the DSA conversion function could enhance[8] the security of DSA. That our proofs fail for a one-way f but work for an almost-invertible f therefore seems counterintuitive.

To reconcile the seeming paradox, we offer the following. Because an almost-invertible function is nearly an identity function—indeed, the identity function is the best example of an almost invertible function—and an identity function clearly cannot add complexity to the design of a protocol, an almost-invertible functions does not add any essential complexity to a protocol. Thus DSA is more complicated than ECDSA because its conversion function departs more from being an identity function. An attack may lie hidden in this complexity until proved otherwise.

That we have a security proof for ECDSA and not DSA because of almost-invertibility reflects (i) our inability to devise a security proof based on the almost-bijectivity and one-wayness conditions on the conversion function, or (ii) the need for stronger conditions than than these two because of some as yet unknown attack that would arise from the failure of this stronger condition. We do know that almost-bijectivity is necessary (§6) and that one-wayness is not necessary (§8 under certain conditions. We do not know whether one-wayness and almost-bijectivity are sufficient.

2.5 Generic Groups

A generic model of a group was first described by Nechaev [41]. Shoup [48] later improved these results and first applied this model to cryptology. Jakobsson and Schnorr [29] also applied a version of the generic model effectively in their security proofs. It has also been used by Abdalla, Bellare and Rogaway [1]. Fischlin [22] points out some pitfalls of the generic group model. The work of Maurer and Wolf [34–36], den Boer [19], and Boneh and Lipton [9] on the relationship between the discrete logarithm problem and the Diffie-Hellman problem relates to a notion similar to that of a generic group, namely a black-box field.

In the generic model of a group, roughly speaking, one assumes that the group operation of a secure group $\mathbb{A}_n$ can be performed only by means of an oracle, and one further assumes that the oracle for the group operation is "random" subject to the constraint of giving valid group operations.

The implications of a security proof in the generic group model are as follows. Roughly speaking, such a proof assures the absence of an adversary who behaves generically with respect to the group by only using the group as an oracle. In other words, no adversary is successful for all groups. But conversely, a security proof in the generic group model does not rule out an adversary being successful for a specific group such as a particular elliptic curve group. Also, the limitations to random oracle models noted by Canetti, Goldreich and Halevi [13] might apply to the generic group model: a security proof in the generic group model might be insufficient to assure the security of each instantiation with a specific group with *efficient* group operations. (If the situations is like that for hash functions, this limitation would follow because the space of group with efficiently computable operations is sufficiently small to be efficiently indexable.)

Our variant of the generic group model includes a number of minor changes. We first describe the differences informally, and then give the whole model in more detail. We fix a public set for $\mathbb{A}_n$, but let the group operations be determined by the oracle. We therefore allow arbitrary inputs to the oracle, unlike [48] where the only inputs allowed were previous inputs. This is a minor difference that does not significantly alter the conclusions, but does model the ability to select compressed elliptic curve points with a nearly arbitrary bit string value. Also we add another operation to the oracle, which reveals information in the private group. (This is similar to how ECDSA or DSA signatures consist of pairs in the private group.) We prove in §7 that these differences do not alter the strength of Nechaev's and Shoup's conclusions about the intractability of discrete logarithms in the generic model. Moreover, our version of the model and the resulting lemma have a form that helps to simplify the security proofs in §8.

Comparision of Generic Group to Generic Hash In many respects, the generic group model is similar to the random oracle model for a hash function. Indeed, one might call the latter model, the generic hash model. In the generic hash model one assumes, roughly speaking, that the hash function can only be accessed by means of an oracle, and further that the oracle for the hash function is "random" subject to the constraint of giving a valid hash function. Thus, both the generic group model and the generic hash model involve random oracles. The two models differ in how they use random oracles. It may be the case that a "generic hash model" is embedded in the generic

[8] One-wayness of the DSA conversion function does enhance the "security" in the following unconventional way: it does not seem feasible to recover a public key from a DSA signature. In ECDSA, the public key can be recovered from the message and the signature because the conversion function is almost-invertible. There may be other advantages of the one-wayness of the DSA conversion function, or perhaps this is the only one.

group model in the following sense. Some of the random outputs of the generic group oracle may be thought of as random outputs of an oracle for some kind of hash function. But even if this is so, it would not mean that the generic group model is a stronger condition than the generic hash model, as we have defined the models; the models are incomparable. This is because the fact that a generic hash might somehow be embedded in a generic group does not relate to any specific group to the hash function, or how well either is modelled by a generic member of its class. In the case of an instantiation of ECDSA, the generic group model means assuming a strong condition for the elliptic curve group but the generic hash model means assuming a strong condition for SHA-1. When thinking of the generic models as conditions, it should in reference to the specific features being modelled. Until there is a formal comparison between an elliptic curve group and SHA-1, neither condition can be considered stronger.

Canetti, Goldreich and Halevi [13] have pointed out some limitations of the random oracle model for hash functions. Their argument uses deep theorems from computer science and diagonal arguments. Loosely speaking, they designed a contrived digital signature scheme that was secure when the hash function was instantiated by a random oracle, but insecure when the hash funcution was instantiated by any efficient function. The contrived security hole involved revealing the signing key when the input to be signed was a description of the efficient hash function used in the instantiation. As noted above, we suspect that there would be simimlar limitations applying to any "random oracle model", regardless of whether it is hash functions, symmetric ciphers or groups being modelled. In particular, we suspect the limitations would apply to the generic group model. Concretely, an example somewhat analogous to theirs but dealing with the generic group model is as follows. Let ECDSA$'$ be defined as follows. To sign a message, generate an ECDSA signature normally, unless the message is a specification of the elliptic curve domain parameters used to instantiate ECDSA$'$ in which case reveal the private key. Clearly ECDSA$'$ is insecure against adaptive attacks for all instantiations of efficient groups, but should remain as secure as ECDSA when instantiated by a generic group.

3 Secure Hash Functions

In this section we define what it means to be a hash function and briefly describe various definitions of security properties for hash functions. For a more thorough treatment of the security issues surrounding the design of hash functions, see Damgård [17, 18], Menezes, van Oorschot and Vanstone [39], Preneel [45], Simon [50] and Stinson [51, 52]. Some of our definitions are somewhat simplified to reflect the common practice of using a fixed hash function in a signature scheme rather than the goal of designing a family of secure hash functions. For this reason, some of our assumptions, particularly our assumption about collision-resistance, are stronger than the usual definitions [17] about families of hash functions. But these stronger assumptions are unavoidable, because the security of the signature schemes we are studying depend on these assumptions about a single hash function.

We define a *hash function* to be a function $h : \{0,1\}^* \to \mathcal{H}$ where the domain $\{0,1\}^*$ is the set of all bit strings and the range is $\mathcal{H} \subseteq \mathbb{Z}_n$ (where $\mathbb{Z}_n$ is the group of integers modulo n). For example, $h(M) = h_{n,\text{SHA-1}}(M) := \iota(\text{SHA-1}(M)) \mod n$, where $\iota(B)$ denotes the integer whose binary representation is B, is a hash function whose range is the set $\{i \in \mathbb{Z}_n | i = j \mod n \text{ for some } 0 \leq j < 2^{160}\}$ (when clear from context, we might occasionally refer to this function $h_{n,\text{SHA-1}}$ more briefly as SHA-1). Hash functions can be *one-way*, *second-preimage-resistant*, *collision-resistant*, *zero-finder-resistant* or *uniform*, as defined below.

3.1 One-Wayness (Preimage-Resistance)

A probabilistic algorithm $\mathtt{I}_h$ is an $(\epsilon_\mathtt{I}, \tau_\mathtt{I})$-*inverter* of hash function h if $\mathtt{I}_h$ accepts input $e \in_R \mathcal{H}$ and outputs a message $M \in \{0,1\}^*$ in running-time[9] at most $\tau_\mathtt{I}$. Further, $h(M) = e$ with probability at least $\epsilon_\mathtt{I}$ assessed over the random choices of both e and $\mathtt{I}_h$. If no such $\mathtt{I}_h$ exists, then h is *one-way* or *preimage-resistant* of strength $(\epsilon_\mathtt{I}, \tau_\mathtt{I})$.

3.2 Second-Preimage-Resistance

Let $\mathcal{S} \subseteq \{0,1\}^*$. A probabilistic algorithm $\mathtt{S}_h$ is an $(\epsilon_\mathtt{S}, \tau_\mathtt{S}, \mathcal{S})$-*second-preimage-finder* for hash function h if $\mathtt{S}_h$ accepts input $M \in_R \mathcal{S}$ and in running-time at most $\tau_\mathtt{S}$ outputs a message $M' \in \{0,1\}^*$. Further, $M \neq M'$ and $h(M') = h(M)$ with probability at least $\epsilon_\mathtt{S}$ assessed over the random choices of both M and $\mathtt{S}_h$. If no such $\mathtt{S}_h$ exists, then h is *second-preimage-resistant* of strength $(\epsilon_\mathtt{S}, \tau_\mathtt{S}, \mathcal{S})$.

[9] Running-time of an algorithm includes the size of the description of the algorithm including any look-up tables that may be used. Time will be measured in units such that a public group operation (see §2) in $\mathbb{A}_n$ takes 1 unit of time.

3.3 Collision-Resistance

A probabilistic algorithm $\mathtt{C}_h$ is an $(\epsilon_\mathtt{C}, \tau_\mathtt{C})$-*collision-finder* for hash function h if in running-time at most $\tau_\mathtt{C}$ it outputs messages $M, M' \in \{0,1\}^*$. Further, $M \neq M'$ and $h(M') = h(M)$ with probability at least $\epsilon_\mathtt{C}$ assessed over the random choices of $\mathtt{C}_h$. If no such $\mathtt{C}_h$ is known, then h is *collision-resistant* of strength $(\epsilon_\mathtt{C}, \tau_\mathtt{C})$.

Of course, since $\{0,1\}^*$ is infinite and $\mathcal{H}$ is finite, collisions for h do exist. And, since $\mathtt{C}_h$ has no input, a very efficient collision-finder $\mathtt{C}_h$ does exist, namely the algorithm that immediately outputs (M, M') for some fixed collision. Nevertheless, for the hash function SHA-1, the best collision-finders *known*[10] with success probability 1 have running-time about 2^{80}. Thus it is reasonable to say that SHA-1 is collision-resistant of strength roughly $(1, 2^{80})$.

For the purposes of ECDSA and DSA, however, Vaudenay [54] noted that collisions in the hash function $h_{n,\text{SHA-1}}$ are what matters. Some measures are therefore needed to help ensure that the domain parameter n was not selected $n = \text{SHA-1}(M_1) - \text{SHA-1}(M_2)$ by a malicious party. Such measures include selection of n by a verifiably random method and selection of n of a special form. Assuming that verifiable randomness methods are sufficiently secure, such measures should ensure[11] that $h_{n,\text{SHA-1}}$ is collision-resistant of strength near that of SHA-1.

3.4 Zero-Finder-Resistance and Zero-Rarity

The hash value zero has a special role in ECDSA and DSA. The following definitions are usueful in describing this role. It should be emphasized that for $h = h_{n,\text{SHA-1}}$, with $n < 2^{160}$, one concerned with messages M such that $\iota(()\text{SHA-1}(M)) \in \{0, n\}$.

A probabilistic algorithm $\mathtt{Z}_h$ is an $(\epsilon_\mathtt{Z}, \tau_\mathtt{Z})$-*zero-finder* for hash function h if $\mathtt{Z}_h$ outputs in running-time at most $\tau_\mathtt{Z}$ a message $M \in \{0,1\}^*$. Further, $h(M) = 0$ with probability at least $\epsilon_\mathtt{Z}$ assessed over the random choices of $\mathtt{Z}_h$. If no such $\mathtt{Z}_h$ is known, then h is *zero-finder-resistant* of strength $(\epsilon_\mathtt{Z}, \tau_\mathtt{Z})$.

Of course, since $\mathtt{Z}_h$ has no input, if there exists any zero-finder $\mathtt{Z}_h$ at all, then there exists a very efficient $\mathtt{Z}_h$, namely one that outputs M which is some known fixed zero. Nevertheless, for the hash function SHA-1, the best zero-finders $\mathtt{Z}_h$ *known* have running-time about 2^{160}. Further study might lead to one with running-time about 2^{80} using some kind of meet-in-the-middle attack (see §3.6 for more discussion).

A hash function h is *frequently zero* of weakness $(\epsilon, \mathcal{S})$ if the following is true. Assume $\mathcal{S}$ is a distribution or set of messages that can be efficiently sampled. Assume the probability that $h(M) = 0$ when $M \in_R \mathcal{S}$ is at least ϵ. The hash function is *zero-rare* of strength $(\epsilon, \mathcal{S})$ if it is not frequently zero of this weakness.

Clearly, a zero-finder-resistant hash is also zero-rare for all valid $\mathcal{S}$. But a zero-rare hash for a particular $\mathcal{S}$ can fail to be zero-finder-resistant. Thus zero-finder-resistance is a stronger condition than the zero-rarity.

3.5 Uniformity (Smoothness)

Let $h : \{0,1\}^* \to \mathcal{H}$ be a hash function. Let $\mathcal{D} \subseteq \{0,1\}^*$ be such that (i) $e = h(M)$ for $M \in_R \mathcal{D}$ can be efficiently generated[12]; and (ii) for each $e \in \mathcal{H}$, the set $P_e = h^{-1}(e) \cap \mathcal{D}$ is sufficiently large so that, even if P_e is known, the probability $1/|P_e|$ is sufficiently small to make guessing a randomly selected secret element of P_e infeasible. Let $b \in_R \{1,2\}$. A probabilistic algorithm $\mathtt{D}_h$ is an $(\epsilon_\mathtt{D}, \tau_\mathtt{D}, \mathcal{D})$-*distinguisher* for h if $\mathtt{D}_h$ accepts as input $e \in_R \mathcal{H}$ if $b = 1$ or $e = h(M)$ for $M \in_R \mathcal{D}$ if $b = 2$, and if $\mathtt{D}_h$ outputs, in running-time at most $\tau_\mathtt{D}$ a guess $d \in \{1,2\}$. Further, $d = b$ with probability at least $\frac{1}{2} + \epsilon_\mathtt{D}$ assessed over the random choices of b, e and $\mathtt{D}_h$. If no such $\mathtt{D}_h$ exists, then h is *uniform* or *smooth* of strength $(\epsilon_\mathtt{D}, \tau_\mathtt{D}, \mathcal{D})$.

Uniformity (or smoothness) can be exhibited by some very simple "hash" functions that are neither collision-resistant nor one-way. For example, the functions $h : \{0,1\}^* \to \mathbb{Z}_n : x \mapsto \bar{x} \mod n$, where $\bar{x}$ is the integer represented by the bit-string x, is uniform for $\mathcal{D} = \{x : 0 \le \bar{x} < n^2, x_1 = 1\}$. Thus, it is highly plausible that $h_{n,\text{SHA-1}}$ is also very uniform for an appropriate choice of $\mathcal{D}$. In particular, consider a distribution $\mathcal{D}$ as follows. If $n < 2^{160}$, choose

[10] As noted earlier, although it would be preferrable to make an assumption about a family of hash functions so that it is possible to hypothesize the non-existence of collision-finders rather than merely our ignorance of collision-finders, the security of the signature schemes we are studying does depend on a single hash function. Therefore, we choose to adopt this stronger variant of collision-resistance.

[11] Though, as the verifiable randomness may be based on SHA-1 itself, there certainly remains a remote potential that some unsuspected weakness in SHA-1 allows an attack.

[12] For example, this is true if $\mathcal{D}$ can be efficiently sampled and the resulting messages are short enough to allow efficient computation of e.

u from $\mathcal{D}$ by repeatedly selecting x uniformly at random from $\{0,1\}^{2\lfloor \log_2(n) \rfloor}$ until $0 \leq \iota(\text{SHA-1}(x)) \leq n$ and then letting $u = x$. If $n > 2^{160}$, choose u from $\mathcal{D}$ by selecting u uniformly at random from $\{0,1\}^{2\lfloor \log_2(n) \rfloor}$. It seems very plausible that $h_{n,\text{SHA-1}}$ is uniform for this choice of $\mathcal{D}$.

Therefore, the uniformity assumption for $h = h_{n,\text{SHA-1}}$ is a mild and reasonable assumption. Indeed, if h drastically fails to be uniform for all choices $\mathcal{D}$, then it seems likely that h takes values in $\mathcal{H}$ that can be distinguished from random values. This distinguishability suggests a smaller range, and thus somewhat inferior collision-resistance. So, we will henceforth use *uniform* to mean uniform of a strength for ϵ_D as small as needed and τ_D as large as needed.

The pseudorandomness aspect of the above definition of uniformity is similar to a property called "entropy-smoothing" by Shoup [49, page 20]. Schweinberger and Shoup [46] also use entropy-smoothing hash functions.

3.6 Relationships Between Hash Security Properties

If the range of the hash-function is infinite, then a collision-resistant hash function can fail to be one-way. The most obvious example is the identity function. Even when the range is finite, as in the case of useful hash functions, collision-resistant hash functions can fail to be one-way [39, Note 9.20]. But Stinson [52] recently proved that collision-resistance and a property milder than our uniformity property together imply one-wayness[13]. Thus when our security results use collision-resistance and uniformity as hypotheses, one-wayness is an implicit additional hypothesis.

It is not clear that any of collision-resistance, uniformity or one-wayness imply zero-finder-resistance. Therefore, in the security proofs, we explicitly hypothesize this condition, if needed. Conversely, zero-finder-resistance does not seem to help imply any of the other properties, even in combination with some of the remaining properties properties. Given a zero-finder-resistant and collision-resitant hash function h, it can be trivially modified into a hash function h' that is collision-resistant but not zero-finder-resistant by setting $h'(x) = h(x)$ unless $x = x_0 = 11010$. Thus it is possible for zero-finder-resistance to be much lower than collision-resistance. But the harm of such weak zero-finder-resistance is that the message x_0 can be forged (§6). In this case, since x_0 is just a single message, this is not likely to be too harmful. Of course, if the zero-finder is capable of finding more than one zero, it is also a collision-finder, so we can rely on collision-resistance. Thus, zero-finder-resistance is far less important than collision-resitance from a practical standpoint.

4 Private Key Generation

The nature of ECDSA and DSA is such that a signer must generate private keys, at least one long-term private key, and at least one ephemeral private key per message to be signed. Recall that a private key is a value in $\mathbb{Z}_n$. Ideally, all the private keys should be generated using a true random generator that yields values uniformly distributed in $\mathbb{Z}_n$. But in practice, true random number generators are too costly for the frequently generated ephemeral private keys, and therefore a deterministic but pseudorandom number generator is used. A method to generate $k \in \mathbb{Z}_n$ is *pseudorandom* if it generates a distribution of k on $\mathbb{Z}_n$ that is indistinguishable from a truly uniform distribution on $\mathbb{Z}_n$. The deterministic generators rely on a secret state to achieve pseudorandomness. The secret state is updated as needed.

The security proofs here are written in terms truly random private key generation. But clearly the pseudorandom generation of the private keys inherits the same security because otherwise the pseudorandomness would be violated. For further discussion see §6.

5 Unforgeability, ECDSA and DSA

Signature schemes and their security were formally defined in [25]. This definition has been widely adopted [4, 5, 11, 44]. We briefly review this definition.

[13] In particular, a uniform, collision-resistant hash is one-way. To see this directly, suppose there was an inverter for a uniform hash function h. Select a random message $M \in_R \mathcal{D}$ and compute $e = h(M)$. Give e to the hash-inverter. The hash inverter outputs some message M', such that $h(M') = e = h(M)$. Thus, (M', M) is a collision, unless $M' = M$. But it is information-theoretically infeasible to find M from e alone, so $M' \neq M$ and a collision is found. Moreover, because e is indistinguishable from random elements in $\mathcal{H}$, the hash-inverter's probability of reduced cannot be affected by the choice of e or else it would be distinguisher. Thus the collision-finder has the same success rate and efficiency as the hash-inverter.

A *signature scheme* is a triple of algorithms: *key generation*, *signature generation* and *signature verification*. Key generation is probabilistic and produces a public and private key. Signature generation is probabilistic, takes as input a message to be signed and a private key, and outputs a signature. Signature verification is deterministic, takes as input a public key, a message and a signature, and then outputs 'accept' or 'reject'.

Existential Forger Let (Q, d) be a public key/private key pair produced by an invocation of the key generation algorithm. An $(\epsilon_{\mathtt{F}}, \tau_{\mathtt{F}}, q_{\mathtt{F}})$-*forger* F of a signature scheme is a probabilistic algorithm whose input is Q, and whose running time is at most $\tau_{\mathtt{F}}$. The forger F is allowed to select a sequence of messages M_i for $1 \leq i \leq q_{\mathtt{F}}$ and invoke the signature generation algorithm to obtain signatures on these messages under the private key d. The forger is allowed to adapt its choices of M_i according to the signatures it obtains. To finish F outputs a message M (different from the M_i's) and a candidate signature σ on M under public key Q. The forger F succeeds if (M, σ) is accepted by the signature verification algorithm under public key Q. The forger's probability of success is at least $\epsilon_{\mathtt{F}}$, where the probability is assessed over random choices of the key generation algorithm, the forger F, and the signing oracle.

Selective Forger An $\mathcal{S}$-*selective* $(\epsilon_{\mathtt{F}}, \tau_{\mathtt{F}}, q)$-*forger* F is as above except that F has as additional input the message M which must be forged by F, where M is drawn at random from $\mathcal{S}$.

Of course, a selective forger can be easily converted to an existential forger. However, since selective forgery may be more damaging than existential forgery in practice, superior resistance to selective forgery may be desired.

"Adversaries" Not Considered One can define non-standard adversaries such as those that find: an additional signature for an already signed message, an additional public-private key pair for a given message-signature pair [7, 30, 37], attacks in the multi-user setting, and attacks depending on nonstandard modified key generation. We will not consider such adversaries here.

5.1 A Signature Scheme Generalizing ECDSA and DSA

We define a class of digital signature schemes, AbstractDSA, which is general enough to include as specializations both DSA and ECDSA.

AbstractDSA Let n be a prime. Let $\mathbb{A}_n$ be a group (for which we will use additive notation) of order n. Let $G \in \mathbb{A}_n$ have order n. Let $f : \mathbb{A}_n \to \mathbb{Z}_n$ be a conversion function. Let $h : \{0,1\}^* \to \mathbb{Z}_n$ be a hash function.

Key Generation: A *private key* is an integer $d \in_R \mathbb{Z}_n \setminus \{0\}$. The corresponding *public key* is $Q = dG$.

Signature Generation: To *sign* a message M do the following:

1. Select $k \in_R \mathbb{Z}_n \setminus \{0\}$.
2. Compute $R = kG$.
3. Compute $r = f(R)$. If $r = 0$, go back to step 1.
4. Compute $e = h(M)$.
5. Compute $s = k^{-1}(e + dr) \mod n$. If $s = 0$, go back to step 1.
6. The *signature* on M is (r, s).

Signature Verification: To *verify* a signature (r, s) on M with public key Q, check that $r, s \in \mathbb{Z}_n \setminus \{0\}$ and

$$r = f\left(s^{-1}h(M)G + s^{-1}rQ\right). \tag{1}$$

ECDSA AbstractDSA specializes to ECDSA as follows. Let $\mathbb{A}_n$ be a subgroup of an elliptic curve group defined over a finite field generated by a point G of order n. (Certain fields and elliptic curves are recommended by ANSI and NIST.) Let f be defined as $f(R) = x_R \mod n$ where x_R is an integer representation of the x-coordinate of the elliptic curve point R. Let $h = h_{n,\text{SHA-1}}$ as defined in §3.

DSA AbstractDSA specializes to DSA as follows. Let $n \approx 2^{160}$ be a prime. Let $p = tn+1$ be another prime, typically larger with $p \approx 2^{1024}$. Let $\mathbb{A}_n$ be a subgroup of $\mathbb{Z}_p^*$ generated by an element $G \in \mathbb{Z}_p^*$ of multiplicative order n. Let f be defined as $f(R) = (R \mod p) \mod n$. Let $h = h_{n,\text{SHA-1}}$ as defined in §3.

6 Necessary Security Conditions

In the absense of certain assumptions of §2, §3, and §4, the following attacks on AbstractDSA are possible. Therefore, these conditions are necessary to the security of AbstractDSA. The conditions in the hypothesis of provable security result concluding existential unforgeability by adaptive chosen-message attacks must include each of these necessary conditions or stronger variants. Results with milder conclusions need only include an appropriate subset of these conditions.

Intractable Discrete Logarithm Problem If discrete logarithm problem in the public-key group $\mathbb{A}_n$ can be solved in time t with success rate ϵ, then the following algorithm $\mathtt{F}$ is an $\mathcal{S}$-selective $(\epsilon, t, 0)$-forger, for any distribution $\mathcal{S}$ of messages.

1. Compute the discrete logarithm d the public key Q.
2. Sign the message the d as the private key.

The results in this paper hypothesize a stronger condition than this, the generic model of a group.

Almost-Bijective Conversion Function If the conversion function f is ϵ_f-clustered for z_0 then the following algorithm $\mathtt{F}$ is an $\mathcal{S}$-selective $(\epsilon_f, 1, 0)$-forger, for any distribution $\mathcal{S}$ of messages.

1. Choose random $s \in \mathbb{Z}_n$.
2. Output (z_0, s).

We repeat that we do not have a bound on ϵ_f for the DSA conversion function. Thus, there is a remote potential that the DSA conversion function might not be almost-bijective and therefore have a weakness that could allow forgery of DSA. On the hand the ECDSA conversion is at most $8/n$-clustered for the recommended elliptic curves.

The main result in this paper hypothesizes a stronger condition than this, almost-invertibility. One of the other results hypothesizes almost-invertibility and the other hypothesizes almost-bijectivity.

One-Way Hash Function If there exists an $(\epsilon_\mathtt{I}, \tau_\mathtt{I})$-inverter $\mathtt{I}_h$ of hash function h, then the following algorithm $\mathtt{F}$ is an $(\epsilon_\mathtt{F}, \tau_\mathtt{F}, 0)$-forger where $\epsilon_\mathtt{F} = \epsilon_\mathtt{I} \cdot |\mathcal{H}|/n$ and $\tau_\mathtt{F} \approx \tau_\mathtt{I}$.

1. Choose random $x, y \in \mathbb{Z}_n$.
2. Compute $V = xG + yQ$.
3. Compute $r = f(V)$, $s = ry^{-1} \bmod n$ and $e = rxy^{-1} \bmod n$.
4. If $e \notin \mathcal{H}$ then halt with failure.
5. Invoke $\mathtt{I}_h$ at e to obtain message M such that $h(M) = e$.
6. Output forgery (r, s) on M.

One of the minor results of this paper hypothesizes one-waynesss of the hash function, but the main result hypothesizes a stronger condition.

Second-Preimage-Resistant Hash If there exists an $(\epsilon_\mathtt{S}, \tau_\mathtt{S}, \mathcal{S})$-second-preimage-finder $\mathtt{S}_h$ of hash function h, then the following algorithm $\mathtt{F}$ is an $\mathcal{S}$-selective $(\epsilon_\mathtt{F}, \tau_\mathtt{F}, 1)$-forger where $\epsilon_\mathtt{F} = \epsilon_\mathtt{S}$ and $\tau_\mathtt{F} \approx \tau_\mathtt{S}$.

1. Invoke $\mathtt{S}_h$ at $M \in \mathcal{S}$ to obtain message M' such that $M \neq M'$ and $h(M) = h(M')$.
2. Invoke the signature algorithm on M' obtaining signature (r, s).
3. Output forgery (r, s) on M.

One of the minor results of this paper hypothesizes second-preimage-resistance of the hash function, but the main result hypothesizes a stronger condition.

Collision-Resistant Hash If there exists an $(\epsilon_{\mathtt{C}}, \tau_{\mathtt{C}})$-collision-finder $\mathtt{C}_h$ of hash function h, then the following algorithm $\mathtt{F}$ is an $(\epsilon_{\mathtt{F}}, \tau_{\mathtt{F}}, 1)$-forger where $\epsilon_{\mathtt{F}} = \epsilon_{\mathtt{C}}$ and $\tau_{\mathtt{F}} \approx \tau_{\mathtt{C}}$.

1. Invoke $\mathtt{C}_h$ to obtain messages M and M' such that $M \neq M'$ and $h(M) = h(M')$.
2. Invoke the signature algorithm on M' obtaining signature (r, s).
3. Output forgery (r, s) on message M.

The main result hypothesizes collision-resistance.

Zero-Finder-Resistant Hash If there exists an $(\epsilon_{\mathtt{Z}}, \tau_{\mathtt{Z}})$-zero-finder $\mathtt{Z}_h$ of hash function h, then the following algorithm $\mathtt{F}$ is an $(\epsilon_{\mathtt{F}}, \tau_{\mathtt{F}}, 0)$-forger where $\epsilon_{\mathtt{F}} = \epsilon_{\mathtt{Z}}$ and $\tau_{\mathtt{F}} \approx \tau_{\mathtt{Z}}$.

1. Invoke $\mathtt{Z}_h$ to obtain message M such that $h(M) = 0$.
2. Choose arbitrary $t \in \mathbb{Z}_n \setminus \{0\}$.
3. Compute $r = f(tQ)$ and $s = t^{-1} f(tQ)$.
4. Output forgery (r, s) on message M.

All the results hypothesize this condition.

Nonzero r-Value Suppose $r = 0$ was accepted in signature verification and that $f(tG) = 0$ for some known t. Then the following algorithm $\mathtt{F}$ is an $\mathcal{S}$-selective $(1, 1, 0)$-forger for any $\mathcal{S}$.

1. Output forgery $(r, s) = (0, t^{-1} h(M))$.

Johnson and Menezes [30] describe a special case of the above attack. Several of the NIST elliptic curves [21, 47] contain points Z such that $f(Z) = 0$. Moreover, although the curve parameters were generated verifiably at random as evidenced by a seed, the generator G is not generated verifiably at random. Without any evidence to the contrary, it is possible that G was selected as $G = t^{-1} Z$ from some random secret t. In this case, the above attack is possible. Therefore, for these curves, it is advisable to check that $r \neq 0$ during verification or else risk selective forgery by an no-message attack, the severest kind of forgery.

Arithmetically Unbiased k-Value In practice, a true random number generator for k is too inefficient. An efficient, but poorly chosen, pseudorandom number generator can create insecurity. Bellare, Goldwasser and Micciancio [3] showed a linear congruential generator is an insecure way to generate k. Smart and Howgrave-Graham [26] showed that if a certain small number of bits of k were revealed for a certain number of signatures, then the private key d could be recovered. Nguyen and Shparlinski [42] improved this result. Bleichenbacher [8] showed that even less bias in k could be exploited to recover the private key: in the case of DSA, even the method recommended in DSS was vulnerable. These attacks say that the generator for k must avoid certain kinds of bias, but not all kinds of bias. For example, these attacks might not be able to exploit biased generators of the following type: select $k = l \mod n$ where $l \in_R [5n, 10n]$ with a smooth integer factorization. Therefore, until an attack is announced the proves that a pseudorandom generator for k is needed, we collect the above necessary conditions on the generator together under the term *arithmetically unbiased*.

All the results of this paper hypothesize a stronger condition than this, pseudo-randomness in the private-key space.

7 A Variant of the Generic Group Model

In this section, the generic group model [29, 48] is reviewed. But because the model here varies slightly from previous version of the model, the essential result that the solving discrete logarithms in the generic group requires about square root of the group order time is reproved. Also, a slightly different notation from previous notations is used in order to accommodate the slight variations in the model.

The Model Recall the notation for e_G and l_G for scalar mulitplication and the discrete logarithm with respect to a base element G in a ssecure group $\langle G \rangle$. More generally, conisder some one-way isomorphism $\alpha : \mathbb{Z}_n \to \mathbb{A}_n$ from $\mathbb{Z}_n$ to some group $\mathbb{A}_n$. Let $\zeta = \alpha^{-1}$. An example is $\alpha = e_G$. Given α, we may think of the group operation on $\mathbb{A}_n$ being defined by the isomorphism α. In other words, we may think of $\mathbb{A}_n$ as some arbitrary set with a group operation defined as let $A + B = \alpha(\zeta(A) + \zeta(B))$ by the $A, B \in \mathbb{A}_n$. Since computing ζ is intractable, computing the group operation of $\mathbb{A}_n$ in this way is intractable. Of course, in practical secure groups $\mathbb{A}_n$, there is some inefficient means of calculating $A + B$. Interestingly, this means that $\alpha(\zeta(A) + \zeta(B))$ is efficiently computable even though $\zeta(A)$ is not. In other words, the addition operation $+$ reveals something about ζ, but not enough to allow ζ to be computed entirely.

More precisely, there are algorithms for computing ζ using on the $+$ operation itself (given n). These algorithms require roughly $\sqrt{n}$ invocations of the group operations and are effective for all groups because they use the group operation as an oracle, without taking any "shortcuts". Algorithms that only invoke the group operations by means of an oracle, are said to be *generic* with respect to the group. In order to formalize this rigourously, we must precisely define what is meant by "only invoking the group operations through an oracle".

To accomplish this, we consider a generic model of the group. In this model, a generic algorithm can invoke the group operations. To ensure that the generic algorithm does not have any other means to get information about ζ, the generic model is randomized so as to leak minimum information about ζ. In other words, given n and $\mathbb{A}_n$, the generic model secretly selects a random mutually inverse pair of bijections $\alpha : \mathbb{Z}_n \to \mathbb{A}_n$ and $\zeta : \mathbb{A}_n \to \mathbb{Z}_n$. The generic model can take two inputs $A, B \in \mathbb{A}_n$ and generate output $A + B = \alpha(\zeta(A) + \zeta(B))$. The randomization ensures the output $A + B$ reveals only the most miminimal information about ζ. Indeed, if C is some other random unrelated element of $\mathbb{A}_n$, then $A + B$ is not likely to reveal any information about $\zeta(C)$ at all.

The stronger assumption that $\alpha : \mathbb{Z}_n \to \mathbb{A}_n$ is a random bijection rather than just one-way bijection is not necessary for the security of the group, but is a useful assumption for a security analysis of a cryptographic protocol like DSA or ECDSA. A security proof that relies on this stronger assumption reflects on how securely a cryptographic scheme uses a group, but not directly on the security under an instantiation with a specific group $\mathbb{A}_n$. Thus, such a proof should be qualified as a security proof in the *generic group model*.

For the purposes of reductive security proofs, it is convenient to let the oracle determine the actions of α and ζ one element at time rather than all at once. The oracle maintains an internal state of pairs $((A_i, z_i))_{1 \le i \le m}$ such that $\alpha(z_i) = A_i$. Doing so fixes one pairing $\alpha : z_i \mapsto A_i$ at a time while guaranteeing that the portion of α fixed so far is the restriction of a random bijection. The relatively small size of the internal state of the oracle makes the reductive security proofs more meaningful.

The oracle is given in Table 1. It takes three commands: *push*, *subtract* and *hint*. Each command appends to the internal state an element pair (A_{m+1}, z_{m+1}) determined by the oracle according the inputs of the command and some random choices the oracle makes. Further details are given below.

Push Accept argument $A \in \mathbb{A}_n$.

1. Let $A_{m+1} = A$.
2. If $A_{m+1} = A_i$ for some $i \in \{1, \ldots, m\}$ then let $z_{m+1} = z_i$.
3. If $A_{m+1} \notin \{A_1, \ldots, A_m\}$ then choose $z_{m+1} \in_R \mathbb{Z}_n \setminus \{z_1, \ldots, z_m\}$.
4. Respond with A_{m+1}. (Redundant.)

Subtract (No argument)

1. Let $z_{m+1} = (z_{m-1} - z_m) \bmod n$.
2. If $z_{m+1} = z_i$ for some $i \in \{1, \ldots, m\}$ then let $A_{m+1} = A_i$.
3. If $z_{m+1} \notin \{z_1, \ldots, z_m\}$ then choose $A_{m+1} \in_R \mathbb{A}_n \setminus \{A_1, \ldots, A_m\}$.
4. Respond with A_{m+1}

Hint Accept argument $h_{m+1} \in \mathbb{Z}_n \setminus \{0\}$.

1. Choose $z_{m+1} \in_R \mathbb{Z}_n \setminus \{z_1, \ldots, z_m\}$.
2. Choose $A_{m+1} \in_R \mathbb{A}_n \setminus \{A_1, \ldots, A_m\}$.
3. Compute $s_{m+1} = z_{m+1}^{-1}(h_{m+1} z_1 + f(A_{m+1}) z_2) \bmod n$ where $f : \mathbb{A}_n \to \mathbb{Z}_n$ is a certain fixed function.
4. Respond with A_{m+1} and s_{m+1}.

Table 1. Generic group oracle commands

Push Command The argument $A \in \mathbb{A}_n$ specifyies A_{m+1} and can be arbitrary. A redundant but convenient output equal to the input is generated. The private value z_{m+1} corresponding to A_{m+1} is selected at random by the oracle, unless it is forced to equal one of the previous private values.

In the security proofs (§8) we shall assume that the first two commands to the oracle are push commands and we let the base generator be $G = A_1$ and the public key to forge be $Q = A_2$. The forger may select A_1 and A_2 arbitrarily provided only that $A_1 \neq A_2$ (because otherwise forgery is trivial). The forger may submit subsequent push commands with arbitrary arguments in $\mathbb{A}_n$.

Subtract Command This takes the previous two element pairs and subtracts them. The output is thus the public value $A_{m+1} = A_{m-1} - A_m$ and the corresponding private value is $z_{m+1} = z_{m-1} - z_m$. The private value z_{m+1} is computed first so that it can be used to determine if A_{m+1} is a previously occurring public value or a new value. If new, A_{m+1} is selected at random from the still available choices for public values.

Other group operations, such as the the addition binary operation, unary operation negation, and the nullary operation zero may be computed by invoking the just a few commands. For example, to find $A + B$, push A twice, subtract to find Z, push A again, subtract to find $-A$, then push B and push $-A$ and then subtract to get $A+B$, using 5 pushes and 3 subtracts. More complicated operations such as scalar multiplication and halving, can be performed with a number of commands proportional to $\lfloor \log_2 n \rfloor$. When modelling certain secure groups where some particular scalar multiplication can be computed faster than above, one should scale the running times by this speedup when applying results in the generic group model.

Hint Command This has an argument h_{m+1}. It also includes certain extra information in the response, essentially the s part of a signature of message whose hash value would be h_{m+1}. The details are given in Table 1.

The hint command of the oracle adds a signing oracle to the group oracle, because $(f(A_{m+1}), s)$ is a signature (§5.1) of any message with hash value h. We note also that the hint command is actually stronger than a signing oracle because it is not necessary to supply an actual message, so that arbitrary hashes can be signed. In the security proofs, the forger does not get direct access to the hint command. Instead, the hint command will be invoked when the forger makes a signing query. We allow the adversary to observe the results of the hint commands. Although only a formality because it is not directly used by the adversary, the hint command is a convenient way to express the way signature reveal information in the private key group (signatures) without leaking additional information, as proved in the lemma below.

DSA and Restricted Push Commands When modelling a DSA subgroup $\langle g \rangle$ it is reasonable to restrict the push commands. A severe restriction would be to insist that all push arguments are equal to previously seen public values, except for the first to two push arguments (the generator $A_1 = g$ and the public key $A_2 = y$ to be forged). A less severe restirction would be to insist that all push arguments are equal to previously seen public values or else are selected at random by the oracle itself. The reason for making such restrictions is that it does not seem possible to find arbitrary elements of the DSA group except by using the supergroup operations. It is possible to use supergroup operations to find an element without using the group operations, but it does not seem possible to control the resulting group element in any way.

In the case of an elliptic curve group, a valid elliptic curve point with an almost arbitrary x-coordinate can be found by solving a quadratic equation. Thus it is reasonable to allow many push commands with nearly arbitrary arguments. Since we wish to consider ECDSA, we will not restrict[14] push commands.

Observable Information In the generic group model, every pair (A_i, z_i) is either *basic* or *derived*. A pair (A_i, z_i) is *basic* if it results from a push command where A_i is distinct from all A_j with $j < i$. All other pairs are *derived*. The oracle reveals sufficient information to determine each derived private element z_i as a $\mathbb{Z}_n$-linear combination of previous basic pairs. To formalize this, we find the following conventions convenient. Let B be the set of indices i for which (A_i, z_i) is basic. In Table 2, we define *coefficients* $C_{ij} \in \mathbb{Z}_n$ for $i \in \{1, \ldots, m\}$ and $j \in B$, such that $z_i = \sum_{j \in B} C_{ij} z_j$. An implicit index m for B and C_{ij} is omitted.

Note that B and C_{ij} are public values that can be determined without knowledge of the private elements and, moreover, can be somewhat controlled through the choice of commands and arguments. Conversely, given B and

[14] Because it is reasonable to model the DSA group by a more restrictive push command, it *might* be the case that the DSA group is more generic and thus more secure in some settings.

Push	1. If $A_{m+1} = A_i$ for some $i \in \{1, \ldots, m\}$ then: (a) Let i be the least such index. (b) Let $C_{(m+1)j} = C_{ij}$ for all $j \in B$. 2. If $A_{m+1} \notin \{A_1, \ldots, A_m\}$ then: (a) Add index $m+1$ to the set B. (b) Let $C_{i(m+1)} = 0$ for all $i \in \{1, \ldots, m\}$. (c) Let $C_{(m+1)(m+1)} = 1$.
Subtract	1. Let $C_{(m+1)j} = C_{mj} - C_{(m-1)j}$ for all $j \in B$.
Hint	1. Let $C_{(m+1)1} = s_{m+1}^{-1} h_{m+1}$. 2. Let $C_{(m+1)2} = s_{m+1}^{-1} f(A_{m+1})$. 3. Let $C_{(m+1)j} = 0$ for all $j \in B \setminus \{1, 2\}$.

Table 2. Derivations

C_{ij}, the random private elements $(z_j)_{j\in B}$ can vary considerably. To formalize this variability, we introduce some more definitions. Let $A = (A_1, \ldots, A_m)$. Let C denote the *coefficient matrix* (C_{ij}) over $\mathbb{Z}_n$, where $i \in \{1, \ldots, m\}$ and $j \in B$. Let S denote the responses to the hint commands. The *basic vector* is $z_B = (z_j)_{j\in B}$ and the *private vector* is $z = (z_i)_{1\le i\le m}$. We have $z = Cz_B$. The row vector $\Delta_i = (C_{ij})_{j\in B}$ is the *derivation* of the pair (A_i, z_i). The derivation of a basic pair (A_j, z_j) will be an elementary row vector with all zeroes except for a single one in the j position.

A *coincidence* at i has occurred in (A, B, C, S) if $A_i = A_j$ but $\Delta_i \ne \Delta_j$ for some $1 \le j < i \le m$. In other words, a coincidence is when one observes public value with two distinct derivations. We are *coincidence-free* if there are no coincidences at any j.

The *basic possibility space* is the set $U_m \subset \mathbb{Z}_n^{|B|}$ of all possible basic vectors z_B consistent with the existing sequence of m commands to the oracle including all the arguments and responses. The *basic proportion* is the fraction $u_m = |Z_m|/n^{|B|}$. The basic proportion shrinks with each oracle response. In the case of coincidence it decreases dramatically, where $u_{m+1} \approx \frac{1}{n} u_m$, because the coincidence reveals a surprising equation and thus a loss of degree of freedom in the basic vector. When the response gives a new derivation with a new value, the basic proportion decreases only slightly, with $u_{m+1} \approx \frac{n-1}{n} u_m$, because the new value only reveals the expected non-equation and thus just a minute loss of freedom.

Unobservable Information Informally, the following result says that for $m \ll \sqrt{n}$, it is infeasible to determine even a single bit of information about the basic private elements, other than the observable information described above. For example, if A_1 and A_2 are basic, determining the discrete logarithm $d = z_2 z_1^{-1}$ of A_2 to the base A_1 is infeasible. In our setup, this means that determining the private key is infeasible.

The essential part of this result is already known [41, 48]. But the version of the model here differs slightly in a couple respects. It includes the hint command and it permits arbitrary inputs to the push commands. Therefore, the result is restated and reproved. The proof is based on the original proof for the earlier models.

The way the following lemma will be used in the security proofs is by inferring a conincidence from a successful signature forgery. The probability of successful forgery can thereby be bounded.

Lemma 1. *Regardless of the strategy used to select the commands and their arguments, after m commands:*

1. *The basic private elements $(z_j)_{j\in B}$ are uniformly distributed over the basic possibility space U_m.*
2. *If the state is coincidence-free, then the basic proportion satisfies $u_m \ge 1 - \binom{m}{2}/n$.*
3. *The probability of the state being coincidence-free is at least $1 - \binom{m}{2}/n$.*

Proof. 1. Assume by induction that the first condition holds for m pairs. We need to show that the first condition holds for $m+1$. If the command is a subtract, hint or push of an old argument, z_B does not change and $U_{m+1} \subset U_m$. Therefore, z_B remains uniformly distributed in U_{m+1}. If the command is a push of a new argument, z_B is augmented by a new private element z_{m+1} that is uniformly chosen from the set $\mathbb{Z}_n \setminus \{z_1, \ldots, z_m\}$, while $U_{m+1} = U_m \times (\mathbb{Z}_n \setminus \{z_1, \ldots, z_m\})$. It follows that the new z_B is uniformly distributed in U_{m+1}.

2. Assume by induction that the second condition holds for m. If the oracle is also coincidence-free up to $m+1$, then we just need to show that $u_{m+1} \ge u_m - \frac{m}{n}$. If the next command is a push with an old argument, or a

subtract with an old derivation, $U_{m+1} = U_m$ and thus $u_{m+1} = u_m \geq u_m - \frac{m}{n}$. If the next command is a push with a new argument then $u_{m+1} \geq u_m \left(1 - \frac{m}{n}\right) \geq u_m - \frac{m}{n}$. If the next command is a hint or a subtract with a new derivation then $\Delta_{m+1} \neq \Delta_i$ for $i \in \{1, \ldots, m\}$ due to being coincidence-free. The number of solutions to $(\Delta_{m+1} - \Delta_i) z_B = 0$ for each i is at most $n^{|B|-1}$. The lack of conincidence excludes at most $m(n^{|B|-1})$ choices of z_B. Thus $u_{m+1} \geq u_m - \frac{m}{n}$.
3. Assume by induction the oracle is coincidence-free up to m with probability at least $1 - \binom{m}{2}/n$. A coincidence can now occur at $m+1$ with probability at most $\frac{m}{n}/u_m^{-1}$. Thus, the oracle is coincidence-free at $m+1$ with probability at least $\left(1 - \binom{m}{2}/n\right)\left(1 - \frac{m}{n}/u_m\right)$. Using the second condition gives the probability at least $\left(1 - \binom{m}{2}/n\right) - \frac{m}{n} = 1 - \binom{m+1}{2}/n$. □

It should be noted that the above result says that finding discrete logarithms is difficult, even when given access to the hint command, which is a kind of signing oracle. But given direct access to the hint command makes forgery easy. Therefore in the security proofs, the forger does not get access to the hint command. Instead, the hint command is only used to answer the forger's signing queries. Nevertheless, the above lemma ensures the forger is still unable to force a coincidence with greater probability than expected. In the security proofs, it is shown that a successful forgery implies a coincidence.

If the above result did not address the hint command as in [48], then the oracle group would never input or output elements in the private group. But signatures consist of pairs of elements in the private group, so an active adversary must somehow access the private group. It is not immediately clear that a lemma dealing only with public group elements would still apply once information about private group elements is leaked to the forger by the signing oracle. What the above lemma says is that, even if certain information about the private elements is revealed to the forger, the essential conclusion of [48] still holds.

8 Sufficient Security Conditions

In the following security proofs in the generic group model, the forger's goal is to forge a signature for public key Q distinct from base point G and we assume without loss of generality that $A_1 = G$ and $A_2 = Q$. Thus the forger can choose an arbitrary base point and public key but the private key, namely $z_2 z_1^{-1}$ will effectively be determined by the oracle at random from $\mathbb{Z}_n \setminus \{1\}$. (There is the small probability $1/n$ that the private key will be undefined because $z_1 = 0$.)

8.1 Existential Unforgeability Against No-Message Attacks

Theorem 2. *If $\mathtt{F}$ is an $(\epsilon_{\mathtt{F}}, \tau_{\mathtt{F}}, 0)$-forger of AbstractDSA with hash function h and an almost-invertible conversion function f in the generic model for $\mathbb{A}_n$, then there exists an $(\epsilon_{\mathtt{I}}, \tau_{\mathtt{I}})$-inverter $\mathtt{I}_h$ and $(\epsilon_{\mathtt{Z}}, \tau_{\mathtt{Z}})$-zero-finder $\mathtt{Z}_h$ for h with*

$$\epsilon_{\mathtt{I}} + \frac{\epsilon_{\mathtt{Z}}}{10\tau_{\mathtt{F}}} \geq \frac{\epsilon_{\mathtt{F}}}{10\tau_{\mathtt{F}}} - \frac{\tau_{\mathtt{F}}}{20n}, \quad \textit{and} \quad \tau_{\mathtt{I}}, \tau_{\mathtt{Z}} \leq 2\tau_{\mathtt{F}}. \tag{2}$$

Proof. Suppose $\mathtt{F}$ is an $(\epsilon_{\mathtt{F}}, \tau_{\mathtt{F}}, 0)$-forger in the generic model. We describe an algorithm using $\mathtt{F}$ as a subroutine, that serves as either an $(\epsilon_{\mathtt{I}}, \tau_{\mathtt{I}})$-inverter $\mathtt{I}_h$ or an $(\epsilon_{\mathtt{Z}}, \tau_{\mathtt{Z}})$-zero-finder $\mathtt{Z}_h$ of h.

$\mathtt{I}_h$ (Inverter) or $\mathtt{Z}_h$ (Zero-finder) Input: e (challenge random hash value to invert) Output: M (message such that $h(M) = e$ or $h(M) = 0$ as elements of $\mathbb{Z}_n$) 1. Invoke $\mathtt{F}$ with a simulated oracle (see below). 2. Obtain a message M and its forgery (r, s) from $\mathtt{F}$. 3. Verify the signature (r, s) using the simulated oracle (see below). 4. Output M.

Table 3. Reduction to an inverter or zero-finder

Any method, such as double-and-add, may be used to verify the signature, provided that the simulated generic oracle is used.

The idea of the simulated oracle is that the above algorithm can provide a oracle for F that is indistinguishable from the subtraction oracle of the generic group model. Therefore, the ability of F to produce forgeries will not be affected. The simulated oracle uses the challenge random value e, the ability to invert the conversion function, and the advance knowledge of the derivation to generate one of the derived public values. The details of the simulated oracle are given in Table 4.

1. Choose some time value t between 0 and $\tau_{\mathtt{F}}$ at random.
2. Run the usual oracle for $\mathbb{A}_n$ (that is, F selects push and subtract commands and arguments, but not hint commands, followed by queries needed for a final signature verification) with one following exception:
   (a) For the first subtract command, say command i, requested after time t where
      i. The only nonzero coefficients in the derivation Δ_i are the first two C_{i1} and C_{i2}.
      ii. The coefficient $C_{i1} \neq 0$.
      iii. The private element z_i is distinct from all previous private elements.
   (b) Choose A_i at random from $f^{-1}(eC_{i1}^{-1}C_{i2})$ by using the invertibility of the conversion function f, if possible (halt and admit failure otherwise).

Table 4. Simulated generic group oracle

For the following discussion, we refer to Δ_i above as the *exceptional derivation* and we refer to the commands that meet the required conditions as *qualified*. We now analyze the success of $\mathtt{I}_h$ in inverting h at e.

Indistinguishability of the simulated oracle is due to the randomness of e and the almost-invertibility of f. The exceptional response A_m will be indistinguishable from the response that would have been generated in the generic group model. We therefore conclude that F succeeds with probability at least $\epsilon_{\mathtt{F}}$. Assume now that F succeeds, which has probability $\epsilon_{\mathtt{F}}$.

During verification of the forged signature (r, s) the simulated oracle is used to compute $V = s^{-1}h(M)G + s^{-1}rQ$. Thus, the last public element is $A_m = V$ for some m. By definition of signature verification, $\Delta_m = (s^{-1}h(M), s^{-1}r, 0, \ldots, 0)$. A case analysis follows.

If $h(M) = 0$, then F has found a zero of h, as required for $\mathtt{Z}_h$. Now assume that $h(M) \neq 0$ and thus $C_{m1} \neq 0$. Because the signature is valid, $r \neq 0$, and thus $C_{m2} \neq 0$. Importantly, Δ_m cannot now equal a basic derivation because it has two nonzero coefficients. If Δ_m is a new derivation and z_m is old then there is a coincidence at m. If Δ_m is a new derivation, z_m is new, and Δ_m is non-exceptional, then A_m is chosen at random from $\mathbb{A}_n \setminus \{A_1, \ldots, A_{m-1}\}$ and the probability that $f(A_m) = r$ is at most $10/(n - m)$ because r is determined before the random choice is made. If Δ_m is old, it equals a previous qualified derivation, which has probability at most $1/\tau_{\mathtt{F}}$ of being non-exceptional.

If $\Delta_m = \Delta_i$ for an exceptional derivation Δ_i for $i \leq m$, then according to the simulated oracle $A_i \in f^{-1}(eC_{i1}^{-1}C_{i2})$. Hence $r = f(A_m) = f(A_i) = eC_{i1}^{-1}C_{i2} = eC_{m1}^{-1}C_{m2} = eh(M)^{-1}r$. Canceling $r \neq 0$ gives $h(M) = e$ as elements of $\mathbb{Z}_n$, as required for $\mathtt{I}_h$. Thus:

$$\epsilon_{\mathtt{I}} \geq \frac{\epsilon_{\mathtt{F}} - \epsilon_{\mathtt{Z}} - \frac{1}{n}\binom{m}{2} - \frac{10}{n-m}}{10\tau_{\mathtt{F}}}, \tag{3}$$

which is computed first by subtracting the probability of finding a zero, hitting a coincidence and choosing a random preimage of r from the probability of successful forgery, and then by multiplication with the probability that f can be successfully inverted in the exceptional derivation (at least 1/10) and with the probability that the Δ_m, if old, equals the exceptional derivation (at least $1/\tau_{\mathtt{F}}$). To get the inequality (2), we use $m \leq \tau_F$.

The running-time of $\mathtt{I}_h$ and $\mathtt{Z}_h$ are roughly the same of F, requiring additional time to perform operations in $\mathbb{Z}_n$ to simulate the oracle for F. In other words, each operation that F needs to perform in $\mathbb{A}_n$ requires just a few operations in $\mathbb{Z}_n$, indeed, little more work than the usual oracle. Also, we may regard operations in $\mathbb{Z}_n$ as much faster than those in $\mathbb{A}_n$. For these reasons, we conclude that $\tau_{\mathtt{I}}, \tau_{\mathtt{Z}} \leq 2\tau_{\mathtt{F}}$. □

8.2 Selective Unforgeability Against Adaptive Chosen-Message Attacks

Theorem 3. *If there exists an $\mathcal{S}$-selective $(\epsilon_{\mathtt{F}}, \tau_{\mathtt{F}}, q_{\mathtt{F}})$-forger $\mathtt{F}_h$ of AbstractDSA in the generic group model for $\mathbb{A}_n$ with hash function h and with conversion function f almost-bijective of strength at least β, then there exists an*

$(\epsilon_{\mathtt{S}}, \tau_{\mathtt{S}}, \mathcal{S})$-second-preimage-finder $\mathtt{S}_h$ and $(\epsilon_{\mathtt{Z}}, \tau_{\mathtt{Z}})$-zero-finder $\mathtt{Z}_h$ where

$$\epsilon_{\mathtt{S}} \geq \epsilon_{\mathtt{F}} - \frac{\tau_{\mathtt{F}}^2}{2n} - \left(\frac{1}{\tau_{\mathtt{F}}} - \frac{1}{n}\right)^{-1}\beta, \quad \textit{and} \quad \tau_{\mathtt{S}} \leq 2\tau_{\mathtt{F}}. \tag{4}$$

Proof. Suppose $\mathtt{F}$ is an $(\epsilon_{\mathtt{F}}, \tau_{\mathtt{F}}, q_{\mathtt{F}})$-forger in the generic group model. We describe an $(\epsilon_{\mathtt{S}}, \tau_{\mathtt{S}})$-second-preimage-finder $\mathtt{S}_h$ of the hash function h that invokes $\mathtt{F}$ as sub-routine.

1. Run the usual oracle for $\mathbb{A}_n$ (where $\mathtt{F}$ decides the queries and learns the responses, followed by queries needed for a final signature verification) with the following exceptions arising from the signing queries of $\mathtt{F}$.
2. To answer a signing queries of $\mathtt{F}$ during command i for a message M_i,
   (a) Give a hint command with argument $h(M_i)$ to the oracle,
   (b) Respond to the signing query with $(f(A_i), s_i)$.

Table 5. Reduction to second-preimage-finder

Any method, such as double-and-add, may be used to verify the signature, provided that the oracle is used. The details of coordinating the signing and oracle follow.

We now analyze the success of $\mathtt{S}_h$ at finding a second preimage M' under h of M. We assume henceforth that $\mathtt{F}$ succeeds, which has probability $\epsilon_{\mathtt{F}}$.

During verification of the forged signature (r, s) the generic oracle is used to compute $V = s^{-1}h(M)G + s^{-1}rQ$. Thus its last response is $A_m = V$ for some m. By definition of signature verification, the last derivation is $\Delta_m = (s^{-1}h(M), s^{-1}r, 0, \ldots, 0)$. One of the following cases is true:

1. Derivation Δ_m equals a previous derivation Δ_i arising from a hint command.
2. Derivation Δ_m equals a previous derivation Δ_i not arising from a hint command.
3. Derivation Δ_m is new but private element z_m is old.
4. Derivation Δ_m is new and private element z_m is new.

In the first case, $\Delta_m = \Delta_i$, which implies that $s^{-1}h(M) = C_{m1} = C_{i1} = s_i^{-1}h(M_i)$ and $s^{-1}r = s^{-1}f(A_m) = C_{m2} = C_{i2} = s_i^{-1}f(A_i)$. Since $A_m = A_i$, we get $s = s_i$ and thus $h(M_i) = h(M)$. In the second case, if $h(M) = 0$ a zero has been found. Otherwise, Δ_i does not equal a basic derivation because $r \neq 0$. Thus A_i is derived, so it is selected by the oracle at random from $\mathbb{A}_n \setminus \{A_1, \ldots, A_{i-1}\}$. But $f(A_i) = f(A_m) = r = C_{m2}C_{m1}^{-1}h(M) = C_{i2}C_{i1}^{-1}h(M)$, which is determined before selection of A_i, thus this has probability at most $n(n-i)^{-1}\beta$ because f is almost-bijective. Summing over all possible i, the probability is at most $mn(n-m)^{-1}\beta$. In the third case, a coincidence has occurred, which has probability at most $\binom{m}{2}/n$. In the fourth case, once again $f(A_m)$ is pre-determined before the random selection of A_m, which has probability of at most $n(n-m)^{-1}\beta$. Thus the probability of success $\epsilon_{\mathtt{S}}$ satisfies:

$$\epsilon_{\mathtt{S}} \geq \epsilon_{\mathtt{F}} - \frac{mn\beta}{n-m} - \frac{1}{n}\binom{m}{2} - \frac{n\beta}{n-m}, \tag{5}$$

which is computed by subtracting the probability of the second, third and fourth cases from the probability of successful forgery. We use $\tau_{\mathtt{F}} < m$ to obtain the desired inequality.

The running-time of $\mathtt{S}_h$ is roughly the same as $\mathtt{F}$, requiring additional time to perform operations in $\mathbb{Z}_n$ to simulate the oracle for $\mathtt{F}$. In other words, each operation that $\mathtt{F}$ needs to perform in $\mathbb{A}_n$ requires just a few operations in $\mathbb{Z}_n$, indeed, little more work than the usual oracle. Also, we may regard operations in $\mathbb{Z}_n$ as much faster than those in $\mathbb{A}_n$. For these reasons, we conclude that $\tau_{\mathtt{S}} \leq 2\tau_{\mathtt{F}}$. □

8.3 Existential Unforgeability Against Adaptive Chosen-Ciphertext Attacks

Theorem 4. *If there exists an $(\epsilon_{\mathtt{F}}, \tau_{\mathtt{F}}, q_{\mathtt{F}})$-forger F_h of AbstractDSA with uniform hash function h and an almost-invertible conversion function f in the generic group model for $\mathbb{A}_n$, then there exists an $(\epsilon_{\mathtt{C}}, \tau_{\mathtt{C}})$-collision-finder $\mathtt{C}_h$ and $(\epsilon_{\mathtt{Z}}, \tau_{\mathtt{Z}})$-zero-finder where*

$$\epsilon_{\mathtt{C}} + \epsilon_{\mathtt{Z}} \geq \epsilon_{\mathtt{F}} - \frac{\tau_{\mathtt{F}}^2}{2n}, \quad \textit{and} \quad \tau_{\mathtt{C}}, \tau_{\mathtt{Z}} \leq 2\tau_{\mathtt{F}}. \tag{6}$$

$\mathtt{C}_h$ (Collision-finder) or $\mathtt{Z}_h$ (Zero-finder)
Output: M, M' (distinct messages such that $h(M) = h(M')$ or $h(M) = 0$ as elements of $\mathbb{Z}_n$)

1. Invoke F with a simulated oracle and signing oracle (see below) using random messages $\hat{M}_i$.
2. Obtain from F, query messages M_i, a forged message M and its forgery (r, s).
3. Verify the signature (r, s) using the simulated oracle (see below).
4. If $h(M) = h(\hat{M}_i)$ let $M' = \hat{M}_i$.
5. If $h(M) = h(M_i)$ let $M' = M_i$.
6. Output messages M, M'.

Table 6. Reduction to collision-finder

Proof. Suppose F is an $(\epsilon_{\mathtt{F}}, \tau_{\mathtt{F}}, q_{\mathtt{F}})$-forger in the generic model. We describe an algorithm using F as a sub-routine, that serves as either an $(\epsilon_{\mathtt{C}}, \tau_{\mathtt{C}})$-collision-finder $\mathtt{C}_h$ or as an $(\epsilon_{\mathtt{Z}}, \tau_{\mathtt{Z}})$-zero-finder $\mathtt{Z}_h$ of h.

Any method, such as double-and-add, may be used to verify the signature, provided that the simulated oracle is used.

The idea is that the above algorithm can provide an indistinguishable simulation of an oracle and signing oracle for F. Thus, the ability of F to produce forgeries will not be affected. The simulated oracle uses random hash values e to generate public values and hints to answer signing queries. The details are given in Figure 7.

1. Treat the oracle subtract command i as usual, with the following exceptions:
   (a) If the following conditions are met:
      i. Derivation Δ_i satisfies $C_{ij} = 0$ for $j \neq 1, 2$.
      ii. Derivation Δ_i satisfies $C_{i1} \neq 0$.
      iii. Private element z_i is new.
   (b) Then do the following:
      i. Choose a random element $\hat{e}_i \in \mathbb{Z}_n$.
      ii. If $\hat{e}_i \in \mathcal{H}$ then
         A. Choose a random message $\hat{M}_i$ from the uniform preimage distribution of h.
         B. Let $\hat{e}_i = h(\hat{M}_i)$ (that is, re-assign the value of $\hat{e}_i$).
      iii. Choose random $A_i \in f^{-1}(\hat{e}_i C_{i1}^{-1} C_{i2})$ if possible (else try again).
2. To answer a signing queries of F during command i for a message M_i,
   (a) Give a hint command with argument $h(M_i)$ to the oracle,
   (b) Respond to the signing query with $(f(A_i), s_i)$.

Table 7. Simulated generic group and signing oracle

For the following discussion, we refer to each modified subtract command and its derivation Δ_i as *exceptional*. We now analyze the success of $\mathtt{C}_h$ and $\mathbb{Z}_h$.

Indistinguishability of the simulated oracle is due to the randomness of $\hat{e}_i$, the uniformity of h and the almost-invertibility of f. We therefore conclude that F succeeds with probability at least $\epsilon_{\mathtt{F}}$. Assume now that F succeeds, which has probability $\epsilon_{\mathtt{F}}$.

During verification of the forged signature (r, s) the simulated oracle is used to compute $V = s^{-1}h(M)G + s^{-1}rQ$. Thus the last response is $A_m = V$ for some m. By definition of signature verification $\Delta_m = (s^{-1}h(M), s^{-1}r, 0, \ldots, 0)$. A case analysis follows.

If $h(M) = 0$, then F has found a zero of h, as required for $\mathtt{Z}_h$. Now assume that $h(M) \neq 0$ and thus $C_{m1} \neq 0$. Because the signature is valid $r \neq 0$, thus $C_{m2} \neq 0$. Importantly, Δ_m cannot now equal a basic derivation because it has two nonzero coefficients. If Δ_m is a new derivation and z_m is old then there is a coincidence at m. If Δ_m is a new derivation, z_m is new, and Δ_m is non-exceptional, then A_m is chosen at random from $\mathbb{A}_n \setminus \{A_1, \ldots, A_{m-1}\}$ and the probability that $f(A_m) = r$ is at most $10/(n - m)$ because r is determined before the random choice is made and f is almost-invertible.

If $\Delta_m = \Delta_i$ for an exceptional derivation Δ_i for $i \leq m$, then according to the simulated oracle $A_i \in f^{-1}(\hat{e}_i C_{i1}^{-1} C_{i2})$. Hence $r = f(A_m) = f(A_i) = \hat{e}_i C_{i1}^{-1} C_{i2} = \hat{e}_i C_{m1}^{-1} C_{m2} = \hat{e}_i h(M)^{-1} r$. Canceling $r \neq 0$ gives $h(M) = \hat{e}_i$ as elements of

$\mathbb{Z}_n$. This clearly means that $\hat{e}_i \in \mathcal{H}$, hence $\hat{e}_i = h(\hat{M}_i)$. This gives a collision because it is infeasible that $M = \hat{M}_i$ because h is uniform and $h(\hat{M}_i)$ does not reveal enough information to determine $\hat{M}_i$.

If $\Delta_m = \Delta_i$ where command i is a hint, then we conclude from $s^{-1}h(M) = C_{m1} = C_{i1} = s_i^{-1}f(M_i)$ and $s^{-1}f(A_m) = C_{m2} = C_{i2} = s_i^{-1}f(A_i)$ that $h(M) = h(M_i)$ since $A_m = A_i$. By definition of a forgery $M \neq M_i$ since signing query messages must be distinct from the forged message.

The probability of finding a collision is thus:

$$\epsilon_{\mathtt{C}} \geq \epsilon_{\mathtt{F}} - \epsilon_{\mathtt{Z}} - \frac{1}{n}\binom{m}{2} - \frac{10}{n-m}, \tag{7}$$

which is computed first by subtracting the probability of finding a zero, hitting a coincidence, and accidentally choosing a random preimage of r, from the probability of successful forgery.

The running-time of $\mathtt{C}_h$ and $\mathtt{Z}_h$ are roughly the same as $\mathtt{F}$, requiring additional time to perform operations in $\mathbb{Z}_n$ to simulate the oracle for $\mathtt{F}$. In other words, each operation that $\mathtt{F}$ needs to perform in $\mathbb{A}_n$ requires just a few operations in $\mathbb{Z}_n$, indeed, little more work than the usual oracle. Also, we may regard operations in $\mathbb{Z}_n$ as much faster than those in $\mathbb{A}_n$. For these reasons, we conclude that $\tau_{\mathtt{C}}, \tau_{\mathtt{Z}} \leq 2\tau_{\mathtt{F}}$. □

9 Conclusion

At present, the best methods to solve the elliptic curve discrete logarithm problem take fully exponential time. Furthermore, the known methods may be regarded as 'generic algorithms' in the sense that they invoke the group operations as a black box. Thus, modeling an elliptic curve group by a generic group is a reasonable paradigm, perhaps as reasonable[15] as modeling a cryptographic hash function by a random oracle.

The hash function SHA-1 is widely considered to be both collision-resistant and one-way. The bound of our security proof for existential forgery of ECDSA against adaptive chosen-message attacks is fairly tight in terms of collision-resistance and zero-finding resistance given our other assumptions. It also seems reasonable that SHA-1 is zero-resistant and uniform, although these properties of SHA-1 are not as well studied as collision-resistance and one-way-ness.

If the DSA conversion function is one-way then Theorems 2 and 4 cannot apply to DSA. Theorem 3 applies to DSA only if one proves or assumes that the DSA conversion function is almost-bijective. The ECDSA conversion is almost-invertible, and thus almost-bijective, so all three theorems apply in this respect.

Acknowledgments

Alfred Menezes, Neal Koblitz, Yongge Wang, Nigel Smart and Doug Stinson were very helpful. Comments from anonymous referees of the Eurocrypt 2001, Crypto 2001 and RSA 2001 program committees lead to further essential improvements. The National Sciences and Engineering Research Council of Canada supported this research with an Industrial Research Fellowship.

References

[1] Abdalla, M., Bellare, M., and Rogaway, P. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *Topics in Cryptology — CT-RSA 2001*, D. Naccache, Ed., vol. 2020 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 143–158.

[2] ANSI X9.62. *Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA).* American National Standards Institute, 1999.

[3] Bellare, M., Goldwasser, S., and Micciancio, D. "Pseudo-Random" number generation within cryptographic algorithms: The DSS case. In *Advances in Cryptology — EUROCRYPT '97*, W. Fumy, Ed., vol. 1233 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 277–291.

[15] Indeed, the roughly n isomorphism classes that can be chosen from for an elliptic curves of a given order n—though far fewer than $n!$ possible bijections $\alpha : \mathbb{Z}_n \to \mathbb{A}_n$—offer some degree of random choice. It is not clear for typical secure hash functions what the corresponding amount choice is.

[4] Bellare, M., and Rogaway, P. The exact security of digital signatures — how to sign with RSA and Rabin. In *Advances in Cryptology — EUROCRYPT '96*, U. Maurer, Ed., vol. 1070 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 399–416.

[5] Bellare, M., and Rogaway, P. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security* (1993), ACM, pp. 62–73.

[6] Blake, I., Seroussi, G., and Smart, N. *Elliptic Curves in Cryptography*, vol. 265 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, Cambridge, 1999.

[7] Blake-Wilson, S., Johnson, D. B., and Menezes, A. J. Key agreement protocols and their security analysis. In *Proceedings of the 6th IMA International Conference on Cryptography and Coding* (1997), vol. 1355 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 30–45.

[8] Bleichenbacher, D. On the generation of one-time keys in DSS. Presented at the Monteverta workshop, 2001.

[9] Boneh, D., and Lipton, R. J. Algorithms for black-box fields and their application to cryptography. In *Advances in Cryptology — CRYPTO '96*, N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, pp. 283–297.

[10] Branstad, D. K., and Smid, M. E. Response to comments on the NIST proposed digital signature standard. In *Advances in Cryptology — CRYPTO '92*, E. F. Brickell, Ed., vol. 740 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 76–88.

[11] Brickell, E. F., Pointcheval, D., Vaudenay, S., and Yung, M. Design validations for discrete logarithm based signature schemes. In *Proceedings of Third International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2000*, H. Imai and Y. Zheng, Eds., vol. 1751 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 276–292.

[12] Brown, D. R. L., and Johnson, D. B. Formal security proofs for a signature scheme with partial message recovery. In *Topics in Cryptology — CT-RSA 2001*, D. Naccache, Ed., vol. 2020 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 126–142.

[13] Canetti, R., Goldreich, O., and Halevi, S. The random oracle methodology, revisited. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing* (1998).

[14] Certicom ECC challenge, Nov. 1997. `http://www.certicom.com/resources/ecc_chall/challenge.html`.

[15] Coron, J.-S. On the exact security of full domain hash. In *Advances in Cryptology — CRYPTO 2000*, M. Bellare, Ed., vol. 1880 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 229–235.

[16] Cramer, R., and Shoup, V. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security 3*, 3 (2000), 161–185. Extended abstract available at `http://www.shoup.net/papers`.

[17] Damgård, I. B. Collision free hash functions and public key signatures schemes. In *Advances in Cryptology — EUROCRYPT '87*, D. Chaum and W. L. Price, Eds., vol. 304 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 203–216.

[18] Damgård, I. B. A design principle for hash functions. In *Advances in Cryptology — CRYPTO '89*, G. Brassard, Ed., vol. 435 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 416–427.

[19] den Boer, B. Diffie-Hellman is a strong as discrete log for certain primes. In *Advances in Cryptology — CRYPTO '88*, S. Goldwasser, Ed., vol. 403 of *Lecture Notes in Computer Science*, Springer-Verlag.

[20] Dwork, C., and Naor, M. An efficient existentially unforgeable signature scheme and its applications. *Journal of Cryptology 11* (1998), 187–208.

[21] FIPS 186-2. *Digital Signature Standard*. National Institute of Standards and Technology, 2000.

[22] Fischlin, M. A note on security proofs in the generic model. In *Advances in Cryptology — ASIACRYPT 2000*, T. Okamoto, Ed., vol. 1976 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 458–469.

[23] Flajolet, P., and Odlyzko, A. M. Random mapping statistics. In *Advances in Cryptology — EUROCRYPT '89*, J.-J. Quisquater and J. Vandewalle, Eds., vol. 434 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 329–354.

[24] Gennaro, R., Halevi, S., and Rabin, T. Secure hash-and-sign without the random oracle. In *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed., vol. 1592 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 123–139.

[25] Goldwasser, S., Micali, S., and Rivest, R. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing 17*, 2 (1998), 281–308.

[26] Howgrave-Graham, N. A., and Smart, N. P. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography 23* (2001), 283–290.

[27] IEEE Std 1363-2000. *Standard Specifications for Public Key Cryptography*. Institute of Electrical and Electronics Engineers, 2000.

[28] ISO/IEC 14888-3. *Information Technology — Security Techniques — Digital Sigantures with Appendix — Part 3: Certificate Based Mechanisms*. International Standards Organization, 1998.

[29] Jakobsson, M., and Schnorr, C. P. Security of signed ElGamal encryption. In *Advances in Cryptology — ASIACRYPT 2000*, T. Okamoto, Ed., vol. 1976 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 73–89. Available at `http://www.mi.informatik.uni-frankfurt.de/research/papers.html`.

[30] Johnson, D., and Menezes, A. The elliptic curve digital signature algorithm (ECDSA). Tech. Rep. CORR 99–34, Deptartment of Combinatorics and Optimization, University of Waterloo, Waterloo, 1999. Available at `http://www.cacr.math.uwaterloo.ca`.

[31] KALISKI, B. S. A pseudo-random bit generator based on elliptic logarithms. In *Advances in Cryptology — CRYPTO '86*, A. M. Odlyzko, Ed., vol. 263 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 84–103.

[32] KOBLITZ, N. Elliptic curve cryptosystems. *Mathematics of Computation 48* (1987), 203–209.

[33] KOBLITZ, N. *Algebraic Aspects of Cryptography*, vol. 3 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 1998.

[34] MAURER, U. Towards the equivalence of breaking the diffie-hellman protocol and computing discrete logarithms. In *Advances in Cryptology — CRYPTO '94*, Y. Desmedt, Ed., vol. 839 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 271–281.

[35] MAURER, U., AND WOLF, S. Lower bounds on generic algorithms in groups. In *Advances in Cryptology — EUROCRYPT '98*, K. Nyberg, Ed., vol. 1403 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 72–84.

[36] MAURER, U., AND WOLF, S. The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms. *SIAM Journal on Computing 28* (1999), 1689–1721.

[37] MENEZES, A., AND SMART, N. Security of signature schemes in a multi-user setting. preprint, 2001.

[38] MENEZES, A. J. *Elliptic Curve Public Key Cryptosystems*. Communications and information theory. Kluwer Academic Press, 1993.

[39] MENEZES, A. J., VAN OORSCHOT, P. C., AND VANSTONE, S. A. *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications. CRC Press, Boca Raton, 1997.

[40] MILLER, V. S. Uses of elliptic curves in cryptography. In *Advances in Cryptology — CRYPTO '85*, H. C. Williams, Ed., vol. 218 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 417–426.

[41] NECHAEV, V. I. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes 55*, 2 (1994), 165–172.

[42] NGUYEN, P. Q., AND SHPARLINSKI, I. E. The insecurity of the digital signature algorithm with partially known nonces. *Journal of Cryptology* (to appear).

[43] OKAMOTO, T. Provably secure and practical identification schemes and corresponding signature schemes. In *Advances in Cryptology — CRYPTO '92*, E. F. Brickell, Ed., vol. 740 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 31–53.

[44] POINTCHEVAL, D., AND STERN, J. Security arguments for digital signatures and blind signatures. *Journal of Cryptology 13*, 3 (2000), 361–396.

[45] PRENEEL, B. The state of cryptographic hash functions. In *Lectures on Data Security*, I. Damgård, Ed., vol. 1561 of *Lecture Notes in Computer Science*, pp. 158–182.

[46] SCHWEINBERGER, T., AND SHOUP, V. ACE: The advanced cryptographic engine. Submission to NESSIE, aug 2000. Available at `http://shoup.net/papers/`.

[47] SEC 1. *Elliptic Curve Cryptography*. Standards for Efficient Cryptography, Available at `www.secg.org`., 2000.

[48] SHOUP, V. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology — EUROCRYPT '97*, W. Fumy, Ed., vol. 1233 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 256–266.

[49] SHOUP, V. A proposal for an ISO standard for public key encryption (version 2.0), Sept. 2001. Available at `http://shoup.net/papers/`.

[50] SIMON, D. R. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In *Advances in Cryptology — EUROCRYPT '98*, K. Nyberg, Ed., vol. 1403 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 334–345.

[51] STINSON, D. R. *Cryptography: Theory and Practice*. Discrete Mathematics and Its Applications. CRC Press, Boca Raton, 1995.

[52] STINSON, D. R. Some observations on the theory of cryptographic hash functions. Cryptology ePrint Archive, Report 2001/020, 2001. Available at `http://eprint.iacr.org/`.

[53] VANSTONE, S. A. Responses to NIST's proposal. *Communications of the ACM 35* (1992), 50–52.

[54] VAUDENAY, S. Hidden collisions on DSS. In *Advances in Cryptology — CRYPTO '96*, N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 83–87.