On the Provable Security of (EC)DSA Signatures

Manuel Fersch
manuel.fersch@rub.de

Eike Kiltz
eike.kiltz@rub.de

Bertram Poettering
bertram.poettering@rub.de

Horst Görtz Institute for IT Security
Ruhr University Bochum, Germany

ABSTRACT

Among the signature schemes most widely deployed in practice are the DSA (Digital Signature Algorithm) and its elliptic curves variant ECDSA. They are represented in many international standards, including IEEE P1363, ANSI X9.62, and FIPS 186-4. Their popularity stands in stark contrast to the absence of rigorous security analyses: Previous works either study modified versions of (EC)DSA or provide a security analysis of unmodified ECDSA in the generic group model. Unfortunately, works following the latter approach assume abstractions of non-algebraic functions over generic groups for which it remains unclear how they translate to the security of ECDSA in practice. For instance, it has been pointed out that prior results in the generic group model actually establish strong unforgeability of ECDSA, a property that the scheme *de facto* does not possess. As, further, no formal results are known for DSA, understanding the security of both schemes remains an open problem.

In this work we propose GenDSA, a signature framework that subsumes both DSA and ECDSA in unmodified form. It carefully models the "modulo q" conversion function of (EC)DSA as a composition of three independent functions. The two outer functions mimic algebraic properties in the function's domain and range, the inner one is modeled as a bijective random oracle. We rigorously prove results on the security of GenDSA that indicate that forging signatures in (EC)DSA is as hard as solving discrete logarithms. Importantly, our proofs do not assume generic group behavior.

Keywords

Provable security; DSA; ECDSA; GOST; SM2

1. INTRODUCTION

Digital signatures.

With the main application of message and entity authentication, digital signature schemes are an omnipresent cryptographic primitive in today's security landscape. The specific schemes deployed most often are the RSA-FDH variant from PKCS#1v1.5, and the DLP-based DSA and ECDSA. For instance, current versions of TLS authenticate servers exclusively on basis of these schemes. Unfortunately, up to now, for none of the schemes a rigorous security analysis is known that would formally establish that the goal of unforgeability is reached (partial results are known, the ones concerning DSA and ECDSA are discussed below). Put differently, currently it is only the (believed) absence of cryptanalytic attacks that indicates the security of the three schemes in practice. By providing new results on DSA and ECDSA this paper aims at relaxing this clearly unsatisfactory situation.

CCS'16, October 24 - 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978413

The Fiat-Shamir transform and the forking lemma.

A classic strategy to construct digital signatures is through the Fiat-Shamir heuristic. A standard example is the scheme by Schnorr which is defined in the discrete logarithm (DLP) setting: in a group $\mathbb{G}$ of prime order q, signing keys x coincide with exponents, verification keys $X = g^x$ are group elements, and a signature on a message m consists of the components $c = H(g^r, m)$ and $s = r + cx \pmod q$. Here, exponent r is freshly picked per signing operation and H is a hash function that maps into the exponent space. Verification works by recovering $g^r = g^s/X^c$ and checking that $c = H(g^r, m)$.

Schnorr signatures are unforgeable if the DLP is hard [20]. The proof crucially relies on modeling H as a programmable random oracle and involves a rewinding technique for extracting the signing key from any successful forger by solving a system of linear equations. The standard vehicle to assess the extraction probability is the *forking lemma* [20].

DSA and ECDSA.

The signature scheme DSA and its close relative ECDSA build on ideas of ElGamal and are defined, like the Schnorr scheme, in the DLP setting. However, they do not result from the Fiat-Shamir heuristic. Indeed, both DSA and ECDSA require two independent hash functions, H and f, that map messages, respectively, group elements into the exponent space $\mathbb{Z}_q$. Function f is also known as the *conversion function*. In a nutshell, if x is a signing key and $X = g^x$ the corresponding verification key, a signature on a message m is a pair $\sigma = (s, t)$ satisfying $t = f(g^r)$ for

$$r = (H(m) + xt)/s \ . \tag{1}$$

Signatures are verified by recovering $g^r = (g^{H(m)}X^t)^{1/s}$ and checking that $f(g^r) = t$. While standards recommend us-

ing (truncated versions of) the hash functions of the SHA family to instantiate H, ad-hoc constructions specific to the discrete logarithm group are used for the conversion function f. Concretely, for DSA, which is defined in a prime-order subgroup of the multiplicative group of some prime field GF(p) and uses the canonic representation of group elements as integers in $[1 \dots p-1]$, function f is defined per $A \mapsto (A \bmod p) \bmod q$. In contrast, ECDSA is defined on elliptic curves over some finite field $\mathbb{F} = \mathrm{GF}(p^n)$ and thus group elements coincide with points $(x, y) \in \mathbb{F} \times \mathbb{F}$; here, mapping $A \mapsto A.x \bmod q$ is used to instantiate f, where $A.x$ denotes an encoding of the x-part of A as an integer. As the quantities p and q in the DSA case, and the integer representation of a GF(p^n) element and q in the ECDSA case, are rather loosely related, the operations $A \bmod p \bmod q$ and $A.x \bmod q$ do not make much sense from an algebraic point of view. Indeed, as our studies suggest, likely the designers of DSA and ECDSA aimed at exploiting specifically the disruptive nature of these functions to achieve security for their schemes.

Note that, although DSA and ECDSA evidently share similarities with the construction of Schnorr, it is not clear at all how to establish their security using the rewinding technique of the forking lemma. Indeed, modeling hash function H as a random oracle and rewinding the adversary to its m-queries does not seem to allow for the extraction of the signing key: If the forger chooses a fresh ephemeral exponent for each forgery, in the corresponding equation system to solve, the number of unknowns always dominates the number of collected equations. Further, modeling f as a random oracle seems prohibitive: For instance, in the case of ECDSA, the result of applying f to a random element in $\mathbb{G}$ is not uniformly distributed in $\mathbb{Z}_q$;[1] even more problematic, finding preimages of the conversion function turns out to be an easy task.[2] As these are properties that a random oracle does not possess, modeling the conversion function as an ideal random function leads to questionable overall results. Put differently, a formal result on the security of DSA/ECDSA that is achieved by replacing f with a random oracle says little about security in practice.[3] Despite the described obstacles, partial results on the security of DSA and ECDSA were established in prior work, as we discuss next.

Prior analyses of DSA and ECDSA.

Brickell *et al.* [2] developed an extended version of the forking lemma [20] and used it to study variants of DSA signatures. In more detail, they defined a general class of signature schemes called *Trusted El Gamal Type Signature Scheme*, and proved all signatures of this type unforgeable in the random oracle model. Special cases of this framework include the scheme DSA-I (reportedly due to Brickell, 1996) in which the conversion function f is replaced by a random oracle, and the scheme DSA-II which is like DSA but with constraint (1) changed to $r = (H(m,t) + xt)/s$, i.e., hash function H is applied to both the message and the ephemeral value $f(g^r)$ (this idea first appeared in [21]). Importantly, their results do not apply to unmodified DSA signatures (nor to ECDSA which the authors do not explicitly mention). In a similar line of work, Malone-Lee and Smart [14] proposed the variants ECDSA-II and ECDSA-III of the ECDSA scheme. Here, with the motivation of strengthening ECDSA against certain anomalies like duplicate signatures (one signature is valid for two messages [24]) and improving the tightness of security reductions, the authors deliberately deviated from the original ECDSA specification. Concluding, all positive results from [21, 2, 14] apply only to modified versions of DSA/ECDSA and thus say little about the signatures as deployed in reality.

Basically all known positive results on *unmodified* ECDSA are due to Brown. In [5, 3] he models the discrete logarithm setting of ECDSA with a *generic group* [22]. While it appears at first that only standard model properties are required of functions H and f, as we explain below it seems that Brown's generic modeling approach for the group *implicitly* also idealizes the conversion function f. Unfortunately, crucial formal aspects of his idealization remain unclear, and thus also the impact of the results on the practical security of ECDSA (and DSA).

The fact that Brown builds on idealized properties of both $\mathbb{G}$ (explicitly) and f (implicitly) indeed causes alarming side-effects *of practical relevance* to emerge. The following issue was noticed by Stern *et al.* [24]. Studying Brown's proof step-by-step reveals that he actually establishes that ECDSA signatures are *strongly unforgeable* (i.e., the adversary cannot forge a fresh signature on a message for which it already knows a signature), in contrast to the fact that it is actually trivial to attack ECDSA in this way.[4] Brown comments on this contradiction in [5, p. 9], saying "that the [generic group] paradigm cannot be stretched to ridiculous lengths" and that any effort to adapt his version of the generic group model to reflect the malleability of signatures would likely "not exclude other properties [...] being contrived that ECDSA does not actually possess but would if modeled by a generic group". Besides the fact that aspects of strong unforgeability actually do play a crucial role in real-life applications like Bitcoin[5] and one should thus be careful with considering them contrived, Brown does not indicate why his results on (plain) unforgeability do not follow from "over-stretching" the generic group model. Thus, the overall question remains open: From a security proof that assumes generic groups behavior, how much can actually be concluded about the *real-world security* of the scheme? Arguably, a satisfactory argument for the unforgeability of ECDSA was not given in [3]. Further, interestingly, Brown's results apply to ECDSA only and apparently cannot be lifted to DSA as the conversion function of the latter seems not to be 'almost-invertible' (as conjectured in [6]). Independently of the findings discussed above, in [6, 4, 3] Brown identifies joint conditions on H, f; some of them are sufficient for the security of ECDSA (but considerably stronger than DLP), others necessary.

[1] On elliptic curves, for only about every second x-value a corresponding curve point exists; this is responsible for a huge bias of the '$x \bmod q$' function.

[2] For DSA this is likely not the case: Inverting its conversion function is conjectured hard in [4].

[3] Technically speaking, *any* proof in the random oracle model has at most heuristic quality. However, modeling the conversion functions of DSA and particularly ECDSA as a random oracle goes far beyond the standard complications.

[4] If (s,t) is a signature on m, then so is $(-s,t)$. This property ultimately has to do with the conversion function ignoring the y-component of its input.

[5] See https://en.bitcoin.it/wiki/Transaction_Malleability

Further results on the security of DSA and ECDSA are by Vaudenay [25, 26], Howgrave-Graham and Smart [9], Nguyen and Shparlinski [18], and Leadbitter *et al.* [13]. Some of these papers either identify or survey conditions that are necessary for the schemes' security, others focus on the robustness of DSA/ECDSA against flaws in implementations and parameter selection.

Proofs in the Generic Group Model.

One can generally question the value of proofs in the generic group model for assessing the security of systems in reality. Reasons include that the model is non-falsifiable, and that certain problems exist that are provably hard in the generic group model yet easy to solve in real-life groups [7]. An independent issue stems from the fact that, while Shoup's generic group model was originally proposed [22] as a tool to study the hardness of *purely algebraic* problems such as the discrete logarithm problem, the conversion function of ECDSA is a non-algebraic component in the sense that it maps group elements to bit-strings without following the operations induced by the group laws. For any corresponding proof framework that replaces group operations by generic versions, it needs to be defined what it means to apply a standard model function to a (generic) group element. While in generic group models that represent group elements with handles (e.g., the one by Maurer [16]) such an operation has to be specified explicitly via oracles, for models that use encodings (e.g., [23, 5, 17, 12]), even if applying functions to corresponding random values might be syntactically well-defined, a closer inspection shows that a corresponding proof implicitly would treat non-algebraic components as *ideal objects*. Concretely, our understanding is that Brown's analyses of ECDSA *implicitly* model the conversion function f as some idealized random object, and that it thus remains unclear which conclusions on the practical security of ECDSA can be drawn from his results. While a reassessment of ECDSA's security in a generic group model with handles might potentially confirm positive results, we are not aware of any such work.[6] The general issues with generic group proofs involving non-algebraic functions are further discussed in Appendix A.

1.1 Our Results

Modeling the conversion function.

We propose GenDSA, an abstract signature framework that subsumes both DSA and ECDSA *in unmodified form*. The scheme is defined relative to a group $\mathbb{G}$ of prime order q, a hash function H, and a conversion function f, and uses a signing procedure compatible with (1). As our main technical contribution, by suitably modeling function f we prove that forging corresponding signatures is as hard as computing discrete logarithms. More precisely, in contrast to idealizing both the group (explicitly) and the conversion function (implicitly, see discussion above), we require generic behaviour exclusively for an inner building block of the conversion function.

Recall that the conversion functions of DSA and ECDSA map group elements in $\mathbb{G}$ to field elements in $\mathbb{Z}_q$ without preserving a visible algebraic structure (the only direct connection between domain and range seems to be the fact $|\mathbb{G}| = |\mathbb{Z}_q|$, but as practical instantiations of f are not bijective, $|f(\mathbb{G})| < |\mathbb{Z}_q|$). Our intuition is that this transformation is at the heart of the schemes' security. Our approach is to model the algebraically disruptive behaviour of f similarly as one models the disordered behavior of cryptographic hash functions with random oracles, but conscientiously taking care that the relevant real-world properties of practical conversion functions are accounted for (e.g., if applicable, their invertibility). More concretely, we decompose conversion function f into three independent functions as per

$$f = \psi \circ \Pi \circ \varphi \qquad \text{(CONVERSION FUNCTION)} \ , \qquad (2)$$

where we require Π to be a bijection. The idea is to reflect in φ the structure of f that involves only its domain and to reflect in ψ the structure that involves only its range; the component that is responsible for disrupting any algebraic link between the domain and the range is modeled by Π. In security proofs we will replace Π by a bijective random oracle (an idealized public bijection that is accessible, in both directions, via oracles; cryptographic constructions that build on such objects include the Even-Mansour cipher and the SHA3 hash function) while assuming standard model properties for $\mathbb{G}$, H, φ, and ψ. We show that DSA and ECDSA can naturally be obtained, together with the specific peculiarities of their conversion functions, by correspondingly instantiating φ and ψ. For instance, in the case of ECDSA, function φ is such that any two elliptic curve points with the same x-component will be mapped to the same element, and function ψ is the "modulo q function" that implements the characteristic bias of the range of f. Note that our choices of φ and ψ are invertible, so that also the composed f is.

The assumption that the concrete functions Π found in (EC)DSA behave like a bijective random oracle may seem quite strong. Indeed, at a bit-level it certainly does not hold: For the Π functions of DSA and ECDSA it is feasible to find inputs x such that x and $\Pi(x)$ are related in some meaningful way. Recall however that the algebraic semantics of the domain and range of Π are crucially different (Π performs the context switch from group $\mathbb{G}$ to field $\mathbb{Z}_q$). That is, even if specific bit patterns travel through Π in unmodified form, what changes is what these patterns mean. Thus, when assessing the applicability of the bijective random oracle paradigm for Π this should be done in conjunction with a close study of its environment, i.e., functions φ and ψ and their respective domains and co-domains. In particular for the settings of DSA and ECDSA we believe that our positive results obtained by idealizing Π serve as a strong indication for the practical security of the schemes. An alternative, more conservative interpretation of our findings would be that they give cryptanalysis a clear direction of persue: If, despite our theorems, DSA and ECDSA turn out to be insecure, then the problem is definitely the interplay of the binary representations of elements of $\mathbb{G}$ and $\mathbb{Z}_q$. Let us finally point out that a bijective random oracle component implicitly also appears in Brown's analyses (see Appendix A for details).

Proving the security of GenDSA.

Consider GenDSA with conversion function $f = \psi \circ \Pi \circ \varphi$ from (2) and assume Π is a bijective random oracle. We prove the existential unforgeability of GenDSA in a modular way. First, in Section 4.1, we show that the notions

[6] Switching from Brown's model to handle-based generic groups is challenging: the crucial part is finding a suitable model for the conversion function.

of existential unforgeability and key-only unforgeability (no signing oracle available) are equivalent for GenDSA, assuming H is collision resistant and φ is semi-injective.[7] Next, in Section 4.2, using the forking lemma for Π we show that the hardness of DLP implies key-only unforgeability of GenDSA, assuming again semi-injectiveness for φ, and ψ-relative division resistance for H (a non-standard yet standard model security property related to ψ, cf. Definition 5). The latter property in particular holds if H is modeled as a random oracle; further, for the specific functions ψ used to instantiate DSA and ECDSA it is implied by plausible standard-model assumptions on H. As a corollary, the named results imply the existential unforgeability of DSA and ECDSA. We note that our reductions are not tight but lose a factor of about Q (the number of bijective random oracle queries) and square the success probabilities. This is standard for forking lemma based analyses and correspondingly holds for most other DLP-based signatures, e.g., Schnorr.

Extension of our results in the full version.

Recall that the results of Brown [5, 3] can be stretched to imply the strong unforgeability of ECDSA, a property that does not hold in practice. We highlight that our arguments immediately fail when trying to stretch them in a similar way (we understand this as supporting our approach). In fact, by pin-pointing where exactly the corresponding proof attempt fails, as a side result we are able to characterize the strong unforgeability conditions of ECDSA and can confirm, for the first time on formal grounds, the common conjecture that the malleability mentioned in Footnote 4 is the only one of ECDSA. In contrast to ECDSA, for DSA we establish strong unforgeability.

In the full version we further propose a more general scheme called GenericElGamal that also covers the standardized signature schemes SM2 [10], GOST [8], and ECGOST [8]. We show that our proof techniques apply also there and establish corresponding unforgeability results. To the best of our knowledge there is no previous security proof for the GOST digital signature standard. For SM2, the only known security evaluation is in the generic group model [27].

2. PRELIMINARIES

We denote random sampling from a finite set $\mathbb{A}$ according to the uniform distribution with $a \leftarrow_{\$} \mathbb{A}$. We use symbol $\leftarrow_{\$}$ also for assignments from randomized algorithms, while we denote assignments from deterministic algorithms and calculations with $\leftarrow$. If q is a prime number, we write $\mathbb{Z}_q$ for the field $\mathbb{Z}/q\mathbb{Z}$ and assume the canonic representation of its elements as a natural number in the interval $[0\,..\,q-1]$. That is, an element $a \in \mathbb{Z}_q$ is invertible iff $a \neq 0$. All algorithms are randomized unless explicitly noted.

Most of our security definitions and proofs use code-based games. Such a game G consists of an Init procedure, one or more procedures to respond to adversary oracle queries, and a Fin procedure. G is executed with an adversary $\mathcal{A}$ as follows: Init is always run first and its outputs are the inputs to $\mathcal{A}$. Next, the oracle queries of $\mathcal{A}$ are answered by the corresponding procedures of G. Finally, $\mathcal{A}$ calls Fin and terminates. Whenever the Stop command is invoked in a game procedure, its argument is considered the output of the game (and the execution of the adversary is halted). By $G^{\mathcal{A}} \Rightarrow \texttt{out}$ we denote the event that G executed with $\mathcal{A}$ invokes Stop with argument out. At the beginning of the game, integer variables are initialized to 0 and set variables to $\emptyset$. When using symbols like $\perp$ we mean special symbols that do not appear as elements of sets.

In games and reductions, when realizing a bijective random oracle Π with domain $\mathbb{A}$ and range $\mathbb{B}$ via lazy sampling, we consider Π interchangeably a partial function $\Pi\colon \mathbb{A} \to \mathbb{B}$, a partial function $\Pi^{-1}\colon \mathbb{B} \to \mathbb{A}$, and a relation $\Pi \subseteq \mathbb{A} \times \mathbb{B}$. In particular, for $(\alpha, \beta) \in \mathbb{A} \times \mathbb{B}$ we denote with $\Pi \leftarrow \Pi \cup \{(\alpha, \beta)\}$ the operation that 'programs' Π such that $\Pi(\alpha) = \beta$ and $\Pi^{-1}(\beta) = \alpha$. We write $\mathrm{Dom}(\Pi)$ and $\mathrm{Rng}(\Pi)$ for the domain and range of Π, i.e., for the sets $\{\alpha \in \mathbb{A} \mid \exists \beta : (\alpha, \beta) \in \Pi\}$ and $\{\beta \in \mathbb{B} \mid \exists \alpha : (\alpha, \beta) \in \Pi\}$, respectively. We write $(\alpha, \cdot) \in \Pi$ and $(\cdot, \beta) \in \Pi$ shorthand for the conditions $\alpha \in \mathrm{Dom}(\Pi)$ and $\beta \in \mathrm{Rng}(\Pi)$.

2.1 Cyclic groups

Definition 1 (Discrete logarithm). *Let $(\mathbb{G}, g, q)$ be a prime-order group, i.e., $\mathbb{G} = \langle g \rangle$ is a cyclic group of prime-order $q = |\mathbb{G}|$. We often write just $\mathbb{G}$ instead of $(\mathbb{G}, g, q)$. We write 1 for the neutral element and let $\mathbb{G}^* = \mathbb{G} \setminus \{1\}$. An inverter algorithm $\mathcal{I}$ is said to (τ, ε)-break the discrete logarithm problem (DLP) in $\mathbb{G}$ if it runs in time at most τ and achieves inversion advantage $\varepsilon = \Pr[x \leftarrow_{\$} \mathbb{Z}_q^*; x' \leftarrow_{\$} \mathcal{I}(g^x) : x' = x]$.*

Definition 2 (Semi-Injective Function). *Let $\mathbb{G}$ be a prime-order group and $\mathbb{A}$ a set. A function $\varphi\colon \mathbb{G}^* \to \mathbb{A}$ is called* semi-injective *if (a) its range $\varphi(\mathbb{G}^*) \subseteq \mathbb{A}$ is efficiently decidable and (b) it is either injective or 2-to-1 with $\varphi(X) = \varphi(Y)$ always implying $Y \in \{X, X^{-1}\}$.*

2.2 Signature schemes

Definition 3 (Signature scheme). *A* signature scheme *consists of algorithms* KGen,Sign,Verify *such that: algorithm* KGen *generates a signing key* sk *and a verification key* pk*; on input a signing key* sk *and a message m algorithm* Sign *generates a signature σ or the failure indicator $\perp$; on input a verification key* pk*, a message m, and a candidate signature σ, deterministic algorithm* Verify *outputs* 0 *or* 1 *to indicate rejection and acceptance, respectively.*

A signature scheme is correct if for all sk, pk, and m, if $\mathsf{Sign}(\mathrm{sk}, m)$ outputs a signature then Verify accepts it.

Definition 4 (Unforgeability). *For a signature scheme, an algorithm $\mathcal{F}$ that interacts in the* CMA *game (cf. Figure 1), runs in at most time τ, poses at most Q_s queries to the* Sign *oracle, and has a forging advantage of $\varepsilon = \Pr[\mathrm{CMA}^{\mathcal{F}} \Rightarrow 1]$, is said to (τ, Q_s, ε)-break the scheme's existential unforgeability under chosen-message attack (euf-cma). Forgers $\mathcal{F}$ that do not query the signing oracle, i.e., have $Q_s = 0$, are referred to as key-only forgers. We denote the corresponding variant of the* CMA *game as* KO.

If the signature scheme is specified in relation to some idealized primitive that is accessed via oracles, we also annotate the maximum number of corresponding queries; for instance, in the random oracle model for a hash function H we use the expression $(\tau, Q_s, Q_H, \varepsilon)$. We always assume that forgers that output a forgery attempt (m^*, σ^*) pose a priori all (public) queries that the verification of σ^* will require.

[7] A function φ is semi-injective if it is 1-to-1 or 2-to-1, and for any two group elements X, Y that collide under φ it holds that $Y \in \{X, X^{-1}\}$.

Procedure INIT	**Procedure** FIN(m^*, σ^*)
00 $(\mathrm{sk}, \mathrm{pk}) \leftarrow_\$ \mathsf{KGen}$	05 If $(m^*, \cdot) \in \mathcal{L}$: Stop with 0
01 Return pk	06 If $\mathsf{Verify}(\mathrm{pk}, m^*, \sigma^*) = 0$:
Procedure SIGN(m)	07 Stop with 0
02 $\sigma \leftarrow_\$ \mathsf{Sign}(\mathrm{sk}, m)$	08 Stop with 1
03 $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, \sigma)\}$	
04 Return σ	

Figure 1: Game CMA for modeling the existential unforgeability of signatures. We refer to the key-only variant of CMA (where no SIGN queries can be posed) as KO.

2.3 Hash functions

For a domain $\mathbb{D}$ and a prime q we consider *implicitly keyed* hash functions $H\colon \mathbb{D} \to \mathbb{Z}_q$. Recall that practical hash functions like MD5 and SHA1 are not explicitly keyed, but instead have an implicit key $k \in K$ (the initialization vector) chosen and fixed by their designer. The experiments of the up-coming security definitions should be understood as implicitly first picking a random key k and giving it to the adversary.

The following definition captures a security property of H relative to a function ψ; it will be our main security requirement on H in the security analysis of GenDSA.

DEFINITION 5 (ψ-DR). *Let $H\colon \mathbb{D} \to \mathbb{Z}_q$ be a hash function, $\mathbb{B}$ a set, and $\psi\colon \mathbb{B} \to \mathbb{Z}_q$ a function. Let $\mathbb{B}^* = \psi^{-1}(\mathbb{Z}_q^*)$. We say $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ $(\tau_{\psi\mathsf{dr}}, \varepsilon_{\psi\mathsf{dr}})$-breaks the* ψ-relative division resistance *(ψ-DR) of H if it runs in at most time $\tau_{\psi\mathsf{dr}}$ and has an advantage of $\varepsilon_{\psi\mathsf{dr}} = \Pr[\beta, \beta' \leftarrow_\$ \mathbb{B}^*; (m, \Gamma) \leftarrow_\$ \mathcal{A}_1(\beta); m' \leftarrow_\$ \mathcal{A}_2(\Gamma, \beta')\colon H(m)/\psi(\beta) = H(m')/\psi(\beta')]$. Here, Γ is some arbitrary state information passed from $\mathcal{A}_1$ to $\mathcal{A}_2$.*

We stress that in this definition $\mathcal{A}$ is not required to output $m \neq m'$. We show in the full version of this paper that in the case of DSA this notion is implied by preimage and zero resistance (cf. [3]) of H. In the case of ECDSA it is implied by a notion called high entropy division resistance, meaning for any t, t' sampled according to a distribution with high entropy, the adversary cannot find m, m' such that $H(m)/t = H(m')/t'$.

2.4 The forking lemma

We reproduce details of the General Forking Lemma [1], but adapt notation to the discrete logarithm setting.

LEMMA 1 (GENERAL FORKING LEMMA). *Fix a group $(\mathbb{G}, g, q)$. Let $Q \geq 1$ be an integer and $\mathbb{B}$ a non-empty finite set. Consider a randomized algorithm A that takes as input an element $X \in \mathbb{G}^*$ and a sequence $\beta_1, \ldots, \beta_Q \in \mathbb{B}$, and either outputs a pair (j, Σ), where $j \in [1\,..\,Q]$ and Σ is an arbitrary value, or aborts outputting a special symbol $\perp$. Associate to A the forking algorithm Fork_A as specified in Figure 2 that takes an element $X \in \mathbb{G}^*$ and either outputs a pair (Σ, Σ') or aborts with $\perp$. Define the* accepting probability *of A as $\mathsf{acc} := \Pr[X \leftarrow_\$ \mathbb{G}^*; \beta_1, \ldots, \beta_Q \leftarrow_\$ \mathbb{B}; \mathsf{A}(X, \beta_1, \ldots, \beta_Q) \not\approx \perp]$ and the probability of* successful forking *as $\mathsf{frk} := \Pr[X \leftarrow_\$ \mathbb{G}^*; \mathsf{Fork}_\mathsf{A}(X) \not\approx \perp]$. Then we have $\mathsf{frk} \geq \mathsf{acc}\,(\mathsf{acc}/Q - 1/|\mathbb{B}|)$.*

$\mathsf{Fork}_\mathsf{A}(X)$
00 Pick random coins ρ for A
01 Pick $\beta_1, \ldots, \beta_Q \leftarrow_\$ \mathbb{B}$
02 $(j, \Sigma) \leftarrow \mathsf{A}(X, \beta_1, \ldots, \beta_Q; \rho)$
03 // If A aborts outputting $\perp$ then also Fork_A aborts
04 Pick $\beta'_j, \ldots, \beta'_Q \leftarrow_\$ \mathbb{B}$
05 Abort if $\beta'_j = \beta_j$
06 $(j', \Sigma') \leftarrow \mathsf{A}(X, \beta_1, \ldots, \beta_{j-1}, \beta'_j, \ldots, \beta'_Q; \rho)$
07 // If A aborts outputting $\perp$ then also Fork_A aborts
08 Abort if $j \neq j'$
09 Output (Σ, Σ')

Figure 2: Forking algorithm Fork_A. If Fork_A aborts in line 03, 05, 07, or 08, then it does so outputting $\perp$.

3. COMMON DSA/ECDSA FRAMEWORK

We first describe the abstract signature scheme GenDSA, and then show that the standardized versions of DSA and ECDSA can be obtained by instantiating the generic components in the right way. Note that Brown [5] uses a similar abstraction, but with a different way to model the conversion function.

DEFINITION 6 (GenDSA FRAMEWORK). *Let $(\mathbb{G}, g, q)$ be a prime-order group, $H\colon \{0,1\}^* \to \mathbb{Z}_q$ be an (implicitly keyed) hash function, and L be an integer. Define the* conversion function $f = \psi \circ \Pi \circ \varphi\colon \mathbb{G}^* \to \mathbb{Z}_q$ *by its components*

$$\mathbb{G}^* \xrightarrow{\varphi} \{0,1\}^L \xrightarrow{\Pi} [0\,..\,2^L - 1] \xrightarrow{\psi} \mathbb{Z}_q \ ,$$

where φ and ψ are efficient functions and Π is an efficient bijection. Then GenDSA is defined by the algorithms of Figure 3.

The conversion function maps elements from group $\mathbb{G}$ to field $\mathbb{Z}_q$. The intuition behind its construction from components φ, Π, ψ is that φ is an (injective) encoding function that deterministically represents abstract group elements as bit strings, ψ is an independent hash function that maps integers from some range to elements of $\mathbb{Z}_q$, and Π is the link in the middle, bridging the range of φ with the domain of ψ. As computer implementations of $\mathbb{G}$'s operations will operate on bit string representations anyway, function φ will often only exist implicitly. In principle, the linking component Π is not required to fulfill any specific property and both DSA and ECDSA instantiate it with function $str2int_L$ (the canonic bijection between bit strings of length L and integers in the range $0\,..\,2^L - 1$). However, not surprisingly, mandating specific properties of φ, ψ will be crucial for the security analysis of GenDSA. Interestingly, as we will see, the injectiveness of φ requested above can be traded for the algebraic property of semi-injectiveness (cf. Definition 2). Indeed, while DSA's φ is injective, ECDSA's φ is actually only semi-injective.

Differences between (EC)DSA and GenDSA.

Before studying the components $\mathbb{G}, H, L, \varphi, \Pi, \psi$ of DSA and ECDSA in detail, we discuss some differences between the general structure of the standardized DSA/ECDSA versions and GenDSA. First, our version of the signing algorithm has the option to abort without outputting a signature. In contrast, signing with standardized DSA and ECDSA never aborts and always outputs a valid signature. Technically, the difference is that in standardized versions

KGen	Sign(x, m)	Verify$(X, m, \langle s, t\rangle)$
00 $x \leftarrow_\$ \mathbb{Z}_q^*$	05 $r \leftarrow_\$ \mathbb{Z}_q$; $R \leftarrow g^r$	14 If $s = 0 \vee t = 0$:
01 $X \leftarrow g^x$	06 If $R = 1$: Return $\perp$	15 Return 0
02 sk $:= x$	07 $t \leftarrow f(R)$	16 $h \leftarrow H(m)$
03 pk $:= X$	08 If $t = 0$: Return $\perp$	17 $U \leftarrow g^h X^t$
04 Return (sk, pk)	09 $h \leftarrow H(m)$	18 If $U = 1$: Return 0
	10 $u \leftarrow h + xt$	19 $R \leftarrow U^{1/s}$
	11 If $u = 0$: Return $\perp$	20 If $f(R) \neq t$:
	12 $s \leftarrow u/r$	21 Return 0
	13 Return $\langle s, t\rangle$	22 Return 1

Figure 3: GenDSA. In line 13, with $\langle\cdot,\cdot\rangle$ we denote any fixed encoding $\mathbb{Z}_q \times \mathbb{Z}_q \to \{0,1\}^*$; correspondingly, in Verify we assume that only signatures with $s, t \in \mathbb{Z}_q$ are considered.

the three "Return $\perp$" statements of our Sign are replaced with "Goto line 05" instructions. That is, effectively, the standardized DSA/ECDSA signing procedures iterate our Sign algorithm until eventually succeeding with outputting a pair $\langle s, t\rangle$. This change clearly does not affect the security assessment of the overall scheme. A second structural difference between standardized DSA/ECDSA and our specification is that in the standards the last instruction of Sign before returning the signature is to assert that $s \neq 0 \wedge t \neq 0$; in our version these conditions are checked as early as possible. As a third change we note that the $U = 1$ abort condition of line 18 is not present in DSA [11] (however, most ECDSA specifications [19] assert that $R \neq 1$ after line 19, which is equivalent to our condition). It is easy to check that adding line 18 neither affects the correctness nor the security of DSA (see the full version for the worked out arguments).

3.1 Instantiations

DSA. We show how to instantiate $\mathbb{G}, H, L, \varphi, \Pi, \psi$ in order to obtain the DSA signature scheme as standardized in [11, Sec. 4] and [19] (modulo the differences already discussed). As the prime-order group $(\mathbb{G}, g, q)$ a subgroup of the multiplicative group of a prime field is used: For security parameters $L \in \{1024, 2048, 3072\}$ and $N \in \{160, 224, 256\}$ (these are the values from [11]), two prime numbers p and q and an element $g \in \mathbb{Z}_p^*$ are chosen such that p has bit-length L, q has bit-length N, $q \mid (p-1)$, $q^2 \nmid (p-1)$, and $\mathrm{ord}_p(g) = q$. The group is then $\mathbb{G} := \langle g\rangle_p \subseteq \mathbb{Z}_p^*$. Further, hash function H is constructed from any hash function of the SHA family specified in FIPS 180-4 that has an output length of at least N bits: The construction is $H(m) := \mathit{str2int}_N(\mathit{sha}(m)) \bmod q$, where $\mathit{sha}\colon \{0,1\}^* \to \{0,1\}^N$ stands for the N leftmost output bits of the chosen hash function. As the conversion function $f\colon \mathbb{G}^* \to \mathbb{Z}_q$ the mapping $x \mapsto x \bmod p \bmod q$ is used (assuming the representation of $\mathbb{G}$'s elements as numbers in $[1\,..\,p-1]$). In the terms of Definition 6 this is achieved by letting function $\varphi\colon \mathbb{G}^* \to \{0,1\}^L$ be the restriction of mapping $[1\,..\,p-1] \to \{0,1\}^L$; $x \mapsto \mathit{int2str}_L(x)$ to the domain $\mathbb{G}^*$ (where $\mathit{int2str}$ is the canonic conversion of an integer to a bit string), defining function $\Pi\colon \{0,1\}^L \to [0\,..\,2^L-1]$ as $x \mapsto \mathit{str2int}_L(x)$, and defining function $\psi\colon [0\,..\,2^L-1] \to \mathbb{Z}_q$ such that $x \mapsto x \bmod q$.

ECDSA. To obtain ECDSA, group $\mathbb{G}$ is instantiated with a prime-order subgroup of the set of points of an elliptic curve defined over some finite field. Specific such curves over prime fields $\mathbb{F} = \mathrm{GF}(p)$ and binary fields $\mathbb{F} = \mathrm{GF}(2^m)$ are recommended by FIPS 186-4 [11]. For both types of curves, the conversion function $f\colon \mathbb{G}^* \to \mathbb{Z}_q$ takes an elliptic curve point $(x, y) \in \mathbb{G}^* \subseteq \mathbb{F} \times \mathbb{F}$, understands $x \in \mathbb{F}$ as an integer, and outputs the value $x \bmod q$. More precisely, if for a prime curve L denotes the bit-length of p, and, for a binary curve we have $L = m$ and further $\mathit{fe2str}\colon \mathbb{F} \to \{0,1\}^L$ is an encoding of field elements as bit strings of length L, then in the terms of Definition 6 the function $\varphi\colon \mathbb{G}^* \to \{0,1\}^L$ is implemented as the mapping $(x, y) \mapsto \mathit{fe2str}(x)$. Bijection $\Pi\colon \{0,1\}^L \to [0\,..\,2^L-1]$, function $\psi\colon [0\,..\,2^L-1] \to \mathbb{Z}_q$, and hash function H are defined as in DSA, i.e., per $x \mapsto \mathit{str2int}_L(x)$ and $x \mapsto x \bmod q$ for the former two and the latter based on a hash function of the SHA family.

Lemma 2. *Let φ be defined as in DSA and ECDSA, respectively. Then φ is semi-injective (cf. Definition 2).*

Proof. In the case of DSA, $\varphi\colon \mathbb{G}^* \to \{0,1\}^L$ is clearly injective. To efficiently check if a given $Z \in \{0,1\}^L$ has a preimage in $\mathbb{G}^*$ under φ, interpret Z as an element in $\mathbb{Z}_p^*$ and accept it if $Z \neq 1$ and $Z^q = 1 \bmod p$. This is a valid membership test for $\mathbb{G}$ because $q \mid (p-1)$ but $q^2 \nmid (p-1)$. This establishes that φ as in DSA is semi-injective.

For elliptic curves over prime or binary fields, for each $x \in \mathbb{F}$ there are at most two solutions $y \in \mathbb{F}$ such that (x, y) is a point on the curve (at least for the Weierstrass curves mandated by the ECDSA standards [19]). If there are two such solutions then the corresponding points $P = (x, y)$ and $Q = (x, y')$ are inverses of each other: $PQ = 1$ (we assume multiplicative notation). Further, if there is only one solution then the corresponding point $P = (x, y)$ has order two and is thus not an element of $\mathbb{G}$ (which is of prime order and thus has no non-trivial subgroup). That is, in the case of ECDSA φ is 2-to-1 and for all $X, Y \in \mathbb{G}^*$ that collide under φ we have either $X = Y$ or $XY = 1$. Furthermore, for all elliptic curves considered in the ECDSA standard, the range of φ is efficiently decidable. Thus, φ is semi-injective. □

4. SECURITY IN THE BRO MODEL

We establish that GenDSA signatures are unforgeable if the DLP is hard in the underlying group, Π behaves like a bijective random oracle, and certain standard model assumptions on H, ψ, and φ are met. We split the proof into two parts: In Section 4.1 we show that for GenDSA the euf-cma notion is equivalent with the key-only notion (i.e., if one can forge GenDSA then one can do so without having access to sample signatures), and in Section 4.2 we show that any key-only forger can be used to break the DLP.

4.1 KO-unforgeability implies euf-cma

The following theorem says that if H is collision resistant, then euf-cma unforgeability of GenDSA is implied by key-only unforgeability, in the bijective random oracle model.

Theorem 1. *Let φ be semi-injective. Let $\mathcal{F}$ be a forger that $(\tau, Q_s, Q_\Pi, \varepsilon)$-breaks euf-cma security of GenDSA. Then, in the bijective random oracle model for Π, there exist a forger $\mathcal{F}_0$ that $(\tau_0, Q_\Pi, \varepsilon_0)$-breaks the KO security of GenDSA and an adversary that $(\tau_{\mathsf{cr}}, \varepsilon_{\mathsf{cr}})$-breaks the collision resistance of H, where*

$$\varepsilon \leq \varepsilon_0 + \varepsilon_{\mathsf{cr}} + \frac{3QQ_s}{(q-1)/2 - Q},\ \tau_0 = \tau + O(Q_s),\ \tau_{\mathsf{cr}} = \tau + O(Q_s).$$

Here, q denotes the group order, Q_s denotes the maximum number of signing queries of $\mathcal{F}$, Q_Π denotes the maximum

number of bijective random oracle queries of $\mathcal{F}$ and $\mathcal{F}_0$, and $Q = Q_s + Q_\Pi$.

We remark that the security statement does not pose any requirement on function ψ. Furthermore, it does not assume the hardness of DLP in group $\mathbb{G}$.

PROOF. We write $\mathbb{A} := \{0,1\}^L$ and $\mathbb{B} := [0\,..\,2^L - 1]$ throughout the proof, i.e., we have functions $\varphi\colon \mathbb{G}^* \to \mathbb{A}$, $\Pi\colon \mathbb{A} \to \mathbb{B}$, and $\psi\colon \mathbb{B} \to \mathbb{Z}_q$. In the bijective random oracle model for Π, fix a forger $\mathcal{F}$ against the euf-cma security of GenDSA that runs in time at most τ and poses at most Q_s signature queries and Q_Π bijective random oracle queries (in the forward or reverse direction). Let $Q = Q_s + Q_\Pi$. The concrete version of the CMA game from Figure 1 with the GenDSA algorithms plugged in is given as **Game $\mathbf{G}_0$** in Figure 4 (excluding lines 18,19). Note that access to bijective random oracle Π is provided via oracles BRO and BRO^{-1}. Our goal is to upper-bound $\varepsilon = \Pr[\text{CMA}^{\mathcal{F}} \Rightarrow 1] = \Pr[G_0^{\mathcal{F}} \Rightarrow 1]$. In line with Definition 4 we assume that $\mathcal{F}$ does not output a forgery attempt without having posed the corresponding Π query first (more precisely, if $\mathcal{F}$ outputs $(m^*, \langle s^*, t^*\rangle)$ and we let $R^* = \left(g^{H(m^*)} X^{t^*}\right)^{1/s^*}$, then either BRO was queried on input $\varphi(R^*)$, or BRO^{-1} was queried such that $\varphi(R^*)$ was the result; see the corresponding remark in line 27).

Procedure INIT
00 Pick $\Pi\colon \mathbb{A} \to \mathbb{B}$
01 $x \leftarrow_\$ \mathbb{Z}_q^*$
02 $X \leftarrow g^x$
03 Return X

Procedure SIGN(m_i)
04 $r_i \leftarrow_\$ \mathbb{Z}_q$; $R_i \leftarrow g^{r_i}$
05 If $R_i = 1$: Return $\perp$
06 $\alpha_i \leftarrow \varphi(R_i)$
07 $\beta_i \leftarrow \Pi(\alpha_i)$
08 $t_i \leftarrow \psi(\beta_i)$
09 If $t_i = 0$: Return $\perp$
10 $h_i \leftarrow H(m_i)$
11 $u_i \leftarrow h_i + x t_i$
12 If $u_i = 0$: Return $\perp$
13 $s_i \leftarrow u_i / r_i$
14 $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m_i, \langle s_i, t_i\rangle)\}$
15 Return $\langle s_i, t_i\rangle$

Procedure BRO(α)
16 Return $\Pi(\alpha)$

Procedure $\text{BRO}^{-1}(\beta)$
17 Return $\Pi^{-1}(\beta)$

Procedure FIN($m^*, \langle s^*, t^*\rangle$)
18 For all $(m, \cdot) \in \mathcal{L}, m \neq m^*$: ($G_1$)
19 If $H(m) = H(m^*)$: Abort (G_1)
20 If $(m^*, \cdot) \in \mathcal{L}$: Abort
21 If $s^* = 0 \vee t^* = 0$: Abort
22 $h^* \leftarrow H(m^*)$
23 $U^* \leftarrow g^{h^*} X^{t^*}$
24 If $U^* = 1$: Abort
25 $R^* \leftarrow (U^*)^{1/s^*}$
26 $\alpha^* \leftarrow \varphi(R^*)$
27 $\beta^* \leftarrow \Pi(\alpha^*)$ // was queried before
28 If $\psi(\beta^*) \neq t^*$: Abort
29 Stop with 1

Figure 4: G_0 and G_1: Game G_0 (excluding lines 18,19) is the CMA game with the GenDSA algorithms plugged in, in the bijective random oracle model for Π. We make explicit the construction $f = \psi \circ \Pi \circ \varphi$ of the conversion function. Game G_1 includes lines 18,19. We write 'Abort' as a shortcut for 'Stop with 0'.

Game $\mathbf{G}_1$ in Figure 4 (including lines 18,19) is the modification of G_0 in which forgeries obtained by finding a collision of hash function H are not counted. By a standard argument we have $\Pr[G_0^{\mathcal{F}} \Rightarrow 1] \leq \Pr[G_1^{\mathcal{F}} \Rightarrow 1] + \varepsilon_{\mathsf{cr}}$ with an adversary that $(\tau_{\mathsf{cr}}, \varepsilon_{\mathsf{cr}})$-breaks the collision resistance of H, where $\tau_{\mathsf{cr}} = \tau + O(Q_s)$.

The intuition behind the overall proof is to use forger $\mathcal{F}$ to construct a key-only forger $\mathcal{F}_0$ that has a similar forging probability as $\mathcal{F}$. More precisely, $\mathcal{F}_0$ shall invoke $\mathcal{F}$, answer all signing and bijective random oracle queries posed by $\mathcal{F}$, and finally forward $\mathcal{F}$'s forgery to its own challenger. Signing queries, as so often in proofs of unforgeability of signature schemes based on the Fiat-Shamir transform, are processed by simulation, which particularly includes programming the Π bijection. More tricky is answering queries to the bijective random oracle: Would $\mathcal{F}_0$ simulate the bijective random oracle all by itself, then also the forgery output by $\mathcal{F}$ would be valid with respect to this instantiation of Π; however, the challenger of $\mathcal{F}_0$ would consider the provided signature invalid, as its own Π implementation would likely disagree. On the other hand, if $\mathcal{F}_0$ would relay all bijective random oracle queries to its own challenger, simulating signatures for $\mathcal{F}$ (i.e., programming the bijective random oracle) would not be possible any more. The solution is to let $\mathcal{F}_0$ combine the two approaches: The bijective random oracle queries that are related to simulated signatures are answered using a local mechanism, all other queries are relayed to the own challenger.

Procedure INIT
00 $(\Pi^O, \Pi^S) \leftarrow (\emptyset, \emptyset)$
01 $x \leftarrow_\$ \mathbb{Z}_q^*$; $X \leftarrow g^x$
02 Return X

Procedure BRO(α)
03 If $(\alpha, \cdot) \in \Pi$: Return $\Pi(\alpha)$
04 $\beta \leftarrow_\$ \mathbb{B} \setminus \text{Rng}(\Pi^O)$
05 If $(\cdot, \beta) \in \Pi^S$: Abort
06 $\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
07 Return β

Procedure SIGN(m_i) (G_2)
08 $r_i \leftarrow_\$ \mathbb{Z}_q$; $R_i \leftarrow g^{r_i}$
09 If $R_i = 1$: Return $\perp$
10 $\alpha_i \leftarrow \varphi(R_i)$
11 If $(\alpha_i, \cdot) \in \Pi$: Abort
12 $\beta_i \leftarrow_\$ \mathbb{B}$
13 If $(\cdot, \beta_i) \in \Pi$: Abort
14 $\Pi^S \leftarrow \Pi^S \cup \{(\alpha_i, \beta_i)\}$
15 $t_i \leftarrow \psi(\beta_i)$
16 If $t_i = 0$: Return $\perp$
17 $h_i \leftarrow H(m_i)$
18 $u_i \leftarrow h_i + x t_i$
19 If $u_i = 0$: Return $\perp$
20 $s_i \leftarrow u_i / r_i$
21 $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m_i, \langle s_i, t_i\rangle)\}$
22 Return $\langle s_i, t_i\rangle$

Procedure $\text{BRO}^{-1}(\beta)$
23 If $(\cdot, \beta) \in \Pi$: Return $\Pi^{-1}(\beta)$
24 $\alpha \leftarrow_\$ \mathbb{A} \setminus \text{Dom}(\Pi^O)$
25 If $(\alpha, \cdot) \in \Pi^S$: Abort
26 $\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
27 Return α

Procedure SIGN(m_i) (G_3)
28 $\beta_i \leftarrow_\$ \mathbb{B}$
29 If $(\cdot, \beta_i) \in \Pi$: Abort
30 $t_i \leftarrow \psi(\beta_i)$
31 If $t_i = 0$: Return $\perp$
32 $h_i \leftarrow H(m_i)$
33 $U_i \leftarrow g^{h_i} X^{t_i}$
34 If $U_i = 1$: Return $\perp$
35 $s_i \leftarrow_\$ \mathbb{Z}_q$
36 If $s_i = 0$: Return $\perp$
37 $R_i \leftarrow U_i^{1/s_i}$
38 $\alpha_i \leftarrow \varphi(R_i)$
39 If $(\alpha_i, \cdot) \in \Pi$: Abort
40 $\Pi^S \leftarrow \Pi^S \cup \{(\alpha_i, \beta_i)\}$
41 $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m_i, \langle s_i, t_i\rangle)\}$
42 Return $\langle s_i, t_i\rangle$

Figure 5: Games G_2 and G_3. We use writing convention $\Pi := \Pi^O \cup \Pi^S$. Procedure FIN is as in G_1 (in Figure 4).

Game $\mathbf{G}_2$ in Figure 5 is a preparation towards the reduction step just described. The major difference to G_1 is that bijection Π is now implemented using lazy sampling. In particular, in G_2 we represent Π as a subset of $\mathbb{A} \times \mathbb{B}$. More precisely, the input-output pairs (α, β) that later shall result from relayed bijective random oracle queries are managed in a list Π^O, the ones originating from programming the bijective random oracle are managed in a list Π^S, and by convention we always have $\Pi := \Pi^O \cup \Pi^S$. In most cases joint list Π will indeed represent a bijection. However, with small probability the Π^O and Π^S parts may become contradictory; whenever this happens, the game aborts (in lines 05, 11, 13, 25). The task of lines 05 and 25 is to guard that $\beta \in \mathbb{B} \setminus \text{Rng}(\Pi)$ and $\alpha \in \mathbb{A} \setminus \text{Dom}(\Pi)$, respectively, and lines 11 and 13 ensure that the independently sampled pairs (α_i, β_i) can be added to Π without causing a conflict. As

$\mathcal{F}$ by assumption does not output a forgery without having posed the corresponding bijective random oracle query first, Procedure FIN is not affected by the switch to lazy sampling and remains unmodified.

We assess the probability that G_2 aborts in lines 05 or 25 as follows: In each case, a uniformly distributed value of a set of cardinality at least $2^L - Q$ is sampled and checked for containedness in a set of at most Q_s elements (elements are added to Π^S only in Procedure SIGN, and at most one element is added per query). That is, per execution of lines 05 and 25 the probability of a hit is at most $Q_s/(2^L - Q)$. As these lines are executed at most Q_Π times in total, the overall probability of abort is bounded by $Q_\Pi Q_s/(2^L - Q)$. Similarly, the probability of a hit in line 13 is at most $Q/2^L$ and thus there the overall probability of abort is bounded by $Q_s Q/2^L$. Consider next the abort condition in line 11. The value α_i is uniformly distributed in a set of cardinality at least $(q-1)/2$ (as R_i is uniform in $\mathbb{G}^*$ and φ is a semi-injective function), and the abort condition is met if this value hits one of the at most Q elements contained in Π. Thus, the overall abort probability for line 11 is bounded by $2Q_sQ/(q-1)$. From the fact that φ is semi-injective we can further deduce that $(q-1)/2 \leq 2^L$. As G_1 and G_2 are identical if no abort happens, all in all we obtain $\Pr[G_1^{\mathcal{F}} \Rightarrow 1] \leq \Pr[G_2^{\mathcal{F}} \Rightarrow 1] + 3QQ_s/((q-1)/2 - Q)$ (see full version for the calculation).

Observe that G_2 uses the signing key in line 18 to implement the signature oracle. We define **Game $\mathbf{G}_3$** (also in Figure 5) like G_2 but with a signature oracle that gets along with only public information. We argue that the forging probability of $\mathcal{F}$ in G_2 and G_3 is identical. To see this, observe that lines 28–34 correspond with lines 12–19 (except for line 14) and induce the same distribution on β_i, t_i, and $u_i = \log U_i$, and that lines 08, 09, 20 and lines 35, 36, 37 correspond and induce the same distribution on $r_i = \log R_i$ and s_i (as in both cases $u_i \neq 0$, and the distribution of u_i is independent of the distribution of r_i and s_i). Thus $\Pr[G_3^{\mathcal{F}} \Rightarrow 1] = \Pr[G_2^{\mathcal{F}} \Rightarrow 1]$.

The code of Figure 6 constructs a forger $\mathcal{F}_0$ from forger $\mathcal{F}$. Importantly, $\mathcal{F}_0$ does not pose signature queries. Assume that $\mathcal{F}_0$ is executed within the KO experiment (cf. Figure 1). Note that $\mathcal{F}_0$ acquires the public key X from its challenger in line 01 and hands it on to $\mathcal{F}$ in line 02. The signature queries posed by $\mathcal{F}$ are processed by $\mathcal{F}_0$ using the code from the simulation in G_3 (which in particular does not require the signing key). There is, though, a difference between how bijective random oracle queries are processed in G_3 and by $\mathcal{F}_0$: While list Π^O in both cases keeps track of input-output pairs that do not originate from programming the bijective random oracle, the sampling steps from lines 04 and 24 of Figure 5 are replaced by queries to the external oracles BRO^* and $(\mathrm{BRO}^*)^{-1}$ in Figure 6, i.e., to the bijective random oracle that is controlled by $\mathcal{F}_0$'s challenger. Clearly, these changes are pure rewriting and not noticeable by $\mathcal{F}$, so the probabilities of reaching the last line of Procedure FIN of G_3 (annotated as line 29 in Figure 4) and reaching line 27 of $\mathcal{F}_0$ in Figure 6 are identical. In the following we argue that this allows concluding that $\Pr[\mathrm{KO}^{\mathcal{F}_0} \Rightarrow 1] = \Pr[G_3^{\mathcal{F}} \Rightarrow 1]$.

Assume that $\mathcal{F}$ delivers a valid forgery $(m^*, \langle s^*, t^* \rangle)$, i.e., that line 27 of $\mathcal{F}_0$ is reached. Let $h^*, R^*, \alpha^*, \beta^*$ be the corresponding values from lines 19, 22, 23, 24. We have $R^* = (g^{h^*} X^{t^*})^{1/s^*}$ and thus $r^* s^* = h^* + x t^*$ for implicit values $r^* = \log R^*$ and $x = \log X$. By assumption either

Procedure INIT
00 $(\Pi^O, \Pi^S) \leftarrow (\emptyset, \emptyset)$
01 $X \leftarrow_\$ \mathrm{INIT}^*$
02 Return X

Procedure $\mathrm{BRO}(\alpha)$
03 If $(\alpha, \cdot) \in \Pi$:
04 Return $\Pi(\alpha)$
05 $\beta \leftarrow \mathrm{BRO}^*(\alpha)$
06 If $(\cdot, \beta) \in \Pi^S$: Abort
07 $\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
08 Return β

Procedure $\mathrm{BRO}^{-1}(\beta)$
09 If $(\cdot, \beta) \in \Pi$:
10 Return $\Pi^{-1}(\beta)$
11 $\alpha \leftarrow (\mathrm{BRO}^*)^{-1}(\beta)$
12 If $(\alpha, \cdot) \in \Pi^S$: Abort
13 $\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
14 Return α

Procedure $\mathrm{FIN}(m^*, \langle s^*, t^* \rangle)$
15 For all $(m, \cdot) \in \mathcal{L}, m \neq m^*$:
16 If $H(m) = H(m^*)$: Abort
17 If $(m^*, \cdot) \in \mathcal{L}$: Abort
18 If $s^* = 0 \vee t^* = 0$: Abort
19 $h^* \leftarrow H(m^*)$
20 $U^* \leftarrow g^{h^*} X^{t^*}$
21 If $U^* = 1$: Abort
22 $R^* \leftarrow (U^*)^{1/s^*}$
23 $\alpha^* \leftarrow \varphi(R^*)$
24 $\beta^* \leftarrow \Pi(\alpha^*)$
25 // was queried before
26 If $\psi(\beta^*) \neq t^*$: Abort
27 If $(\alpha^*, \cdot) \in \Pi^S$:
28 Find $x = \log X$
29 as described in text
30 $\langle s^*, t^* \rangle \leftarrow_\$ \mathsf{Sign}^{\mathrm{BRO}^*}(x, m^*)$
31 Invoke $\mathrm{FIN}^*(m^*, \langle s^*, t^* \rangle)$

Figure 6: Construction of key-only forger $\mathcal{F}_0$ from $\mathcal{F}$. Procedures $\mathrm{INIT}^*, \mathrm{BRO}^*, (\mathrm{BRO}^*)^{-1}$, and FIN^* are the oracles of the KO game in which $\mathcal{F}_0$ is executed. The invocation of the GenDSA signing algorithm in line 30 is relative to bijective random oracle BRO^*. We write 'Abort' as a shortcut for 'Invoke $\mathrm{FIN}^*(\perp, \perp)$'. Procedure SIGN is as in G_3 (cf. Figure 5).

$(\alpha^*, \beta^*) \in \Pi^S$ or $(\alpha^*, \beta^*) \in \Pi^O$. Consider first the case that pair (α^*, β^*) was established in a signature simulation, i.e., $(\alpha^*, \beta^*) \in \Pi^S$. In this case $\mathcal{F}_0$ knows $\alpha_i, \beta_i, R_i, h_i, t_i, s_i$ such that $(\alpha^*, \beta^*) = (\alpha_i, \beta_i)$ and $R_i = (g^{h_i} X^{t_i})^{1/s_i}$, i.e., $r_i s_i = h_i + x t_i$ for implicit $r_i = \log R_i$ and $x = \log X$. We further have $\varphi(R^*) = \alpha^* = \alpha_i = \varphi(R_i)$ and thus $r^* = \eta r_i$ with $\eta \in \{\pm 1\}$ (by the semi-injectiveness of φ, cf. Definition 2), and $t^* = \psi(\beta^*) = \psi(\beta_i) = t_i$. Taken together the above observations result in the linear equation system

$$\begin{pmatrix} \eta s^* & -t_i \\ s_i & -t_i \end{pmatrix} \begin{pmatrix} r_i \\ x \end{pmatrix} = \begin{pmatrix} h^* \\ h_i \end{pmatrix} \tag{3}$$

which, as $t_i \neq 0$, has a unique solution whenever $\eta s^* \neq s_i$. Observe that, as $r_i \neq 0$, case $\eta s^* = s_i$ would imply $h^* = h_i$ and thus $H(m^*) = H(m_i)$, which is excluded by lines 15, 16, 17. Thus $\mathcal{F}_0$ can always recover signing key x, and the signature $\langle s^*, t^* \rangle$ freshly generated in line 30 is valid with respect to oracle BRO^*, i.e., the bijective random oracle managed by the challenger of $\mathcal{F}_0$. That is, if $\mathcal{F}$ forges successfully then so does $\mathcal{F}_0$. Assume next that pair (α^*, β^*) was established by a relayed bijective random oracle query, i.e., $(\alpha^*, \beta^*) \in \Pi^O$. In this case, $\mathcal{F}$'s forgery $(m^*, \langle s^*, t^* \rangle)$ is consistent with oracle BRO^*. Again, if $\mathcal{F}$ forges successfully then so does $\mathcal{F}_0$. We thus obtain $\Pr[\mathrm{KO}^{\mathcal{F}_0} \Rightarrow 1] = \Pr[G_3^{\mathcal{F}} \Rightarrow 1]$, with a forger $\mathcal{F}_0$ that has a running time of at most $\tau_0 = \tau + O(Q_s)$. All in all we proved $\Pr[\mathrm{CMA}^{\mathcal{F}} \Rightarrow 1] \leq \Pr[\mathrm{KO}^{\mathcal{F}_0} \Rightarrow 1] + \varepsilon_{\mathsf{cr}} + 3QQ_s/((q-1)/2 - Q)$, as required. □

4.2 DLP-hardness implies KO-unforgeability

We next show that if the DLP is hard in $\mathbb{G}$ and if H is ψ-relative division resistant, then GenDSA is key-only unforgeable in the bijective random oracle model for Π.

THEOREM 2. *Let φ be semi-injective. Let $\mathcal{F}$ (τ, Q, ε)-break the key-only security of GenDSA. Then if Π is modeled as a bijective random oracle, there exist an adversary $\mathcal{B}$ that $(\tau_{\psi\mathsf{dr}}, \varepsilon_{\psi\mathsf{dr}})$-breaks the ψ-relative division resistance*

of H, and inverters that (τ', ε')-break and (τ'', ε'')-break, respectively, the DLP in $\mathbb{G}$, such that

$$\varepsilon \leq \sqrt{2Q^2\varepsilon_{\psi\mathsf{dr}} + 2Q\varepsilon'} + \varepsilon'' + Q^2/2^L$$

and $\tau_{\psi\mathsf{dr}} = \tau' = 2\tau + O(Q), \tau'' = \tau + O(Q)$. Here, Q denotes a bound on the bijective random oracle queries.

PROOF. We write $\mathbb{A} := \{0,1\}^L$ and $\mathbb{B} := [0\,..\,2^L - 1]$ as in the proof of Theorem 1. In the bijective random oracle model for Π, fix a forger $\mathcal{F}$ against the key-only security of GenDSA that runs in time at most τ and poses at most Q bijective random oracle queries. Following Definition 4 we assume that $\mathcal{F}$ does not output a forgery attempt $(m^*, \langle s^*, t^*\rangle)$ without having posed the corresponding bijective random oracle query first. The concrete version of the KO game from Figure 1 with the GenDSA algorithms plugged in is given as **Game $\mathbf{G}_0$** in Figure 4 (for the current proof, Procedure SIGN and line 20 are redundant and can be ignored). Our goal is to upper-bound forging probability $\varepsilon = \Pr[\mathrm{KO}^{\mathcal{F}} \Rightarrow 1] = \Pr[G_0^{\mathcal{F}} \Rightarrow 1]$.

Game $\mathbf{G}_1$ (in Figure 7) is the modification of G_0 in which bijective random oracle Π is implemented via lazy sampling. Depending on whether they originate from a BRO or a BRO^{-1} query, new input-output pairs (α, β) are stored either in list $\Pi_{\rightarrow}$ or in list $\Pi_{\leftarrow}$ (lines 21 and 30). Taken together, $\Pi_{\rightarrow}$ and $\Pi_{\leftarrow}$ make up list Π, i.e., we assume writing convention $\Pi := \Pi_{\rightarrow} \cup \Pi_{\leftarrow}$. The switch from G_0 to G_1 introduces abort conditions in lines 20 and 29. Inspection shows that each condition is met with probability at most $Q/2^L$ per oracle invocation, so the overall abort probability for these lines is $Q^2/2^L$. As games G_0 and G_1 are identical as long as these conditions are not met, $\Pr[G_0^{\mathcal{F}} \Rightarrow 1] \leq \Pr[G_1^{\mathcal{F}} \Rightarrow 1] + Q^2/2^L$.

Procedure INIT	**Procedure** $\mathrm{BRO}(\alpha)$
00 $(\Pi_{\rightarrow}, \Pi_{\leftarrow}) \leftarrow (\emptyset, \emptyset)$	16 If $(\alpha, \cdot) \in \Pi$:
01 $\beta_1, \ldots, \beta_Q \leftarrow_{\$} \mathbb{B}$ (G_2, G_3)	17 Return $\Pi(\alpha)$
02 $x \leftarrow_{\$} \mathbb{Z}_q^*$; $X \leftarrow g^x$	18 $\beta \leftarrow_{\$} \mathbb{B}$ (G_1)
03 Return X	19 $k \leftarrow k{+}1$; $\beta \leftarrow \beta_k$ (G_2, G_3)
Procedure $\mathrm{FIN}(m^*, \langle s^*, t^*\rangle)$	20 If $(\cdot, \beta) \in \Pi$: Abort
04 If $s^* = 0 \vee t^* = 0$: Abort	21 $\Pi_{\rightarrow} \leftarrow \Pi_{\rightarrow} \cup \{(\alpha, \beta)\}$
05 $U^* \leftarrow g^{H(m^*)} X^{t^*}$	22 Return β
06 If $U^* = 1$: Abort	**Procedure** $\mathrm{BRO}^{-1}(\beta)$
07 $R^* \leftarrow (U^*)^{1/s^*}$	23 If $(\cdot, \beta) \in \Pi$:
08 $\alpha^* \leftarrow \varphi(R^*)$	24 Return $\Pi^{-1}(\beta)$
09 $\beta^* \leftarrow \Pi(\alpha^*)$	25 $\alpha \leftarrow_{\$} \mathbb{A}$
10 // was queried before	26 If $\alpha \in \varphi(\mathbb{G}^*)$: (G_2, G_3)
11 If $\psi(\beta^*) \neq t^*$: Abort	27 $a \leftarrow_{\$} \mathbb{Z}_q^*$ (G_2, G_3)
12 If $(\alpha^*, \beta^*) \in \Pi_{\leftarrow}$: Abort (G_3)	28 $\alpha \leftarrow \varphi(g^a)$ (G_2, G_3)
13 // Note: $\beta^* = \beta_j$ (G_3)	29 If $(\alpha, \cdot) \in \Pi$: Abort
14 // for unique $j \in [1\,..\,k]$ (G_3)	30 $\Pi_{\leftarrow} \leftarrow \Pi_{\leftarrow} \cup \{(\alpha, \beta)\}$
15 Stop with 1	31 Return α

Figure 7: Games G_1, G_2, and G_3. We use writing convention $\Pi := \Pi_{\rightarrow} \cup \Pi_{\leftarrow}$ and write 'Abort' as a shortcut for 'Stop with 0'.

Consider next **Game $\mathbf{G}_2$** from Figure 7. There are two differences to G_1. First, in the BRO oracle, instead of sampling values β individually for each query (line 18), these are now sampled a priori (line 01) and spent one by one (line 19). Second, in the BRO^{-1} oracle, if a sampled value α happens to be an element of $\varphi(\mathbb{G}^*)$ it is resampled such that the discrete logarithm a of a preimage under φ is known (lines 26–28). As φ is a semi-injective function this change does not affect the distribution of sampled values α. We thus have $\Pr[G_1^{\mathcal{F}} \Rightarrow 1] = \Pr[G_2^{\mathcal{F}} \Rightarrow 1]$.

Observe that if $\mathcal{F}$ succeeds in G_2 with finding a valid forgery $(m^*, \langle s^*, t^*\rangle)$ then the corresponding pair (α^*, β^*) (cf. line 09) was established in either a forward or a backward query to the bijective random oracle, i.e., it is contained either in list $\Pi_{\leftarrow}$ or in list $\Pi_{\rightarrow}$. We next prove that from any adversary that forges using a pair established in a backward query we can construct a DLP solver that recovers the discrete logarithm of X with the same advantage. Assume thus that $\mathcal{F}$ forges such that $(\alpha^*, \beta^*) \in \Pi_{\leftarrow}$. Then we have $R^* = (g^{H(m^*)} X^{t^*})^{1/s^*}$, which gives rise to equation $H(m^*) + xt^* = s^* r^*$ with unknowns $x = \log X$ and $r^* = \log R^*$, and we have $\varphi(R^*) = \alpha^*$. By lines 26–28 and the fact that φ is semi-injective we know that $R^* = g^{\eta a}$ for a known a and $\eta \in \{\pm 1\}$. This allows solving above equation for x, as required. The specification of **Game $\mathbf{G}_3$** (in Figure 7) adds to G_2 an abort condition (in line 12) that is triggered whenever $\mathcal{F}$ performs a forgery of the type just considered. A standard reductionist argument thus shows that $\Pr[G_2^{\mathcal{F}} \Rightarrow 1] \leq \Pr[G_3^{\mathcal{F}} \Rightarrow 1] + \varepsilon''$, for an inverter that (τ'', ε'')-breaks DLP in time at most $\tau'' = \tau + O(Q)$.

We employ the forking lemma (cf. Section 2.4) to analyze $\Pr[G_3^{\mathcal{F}} \Rightarrow 1]$, i.e., the probability that $\mathcal{F}$ successfully forges using a forward query (in $\Pi_{\rightarrow}$). To this end, from $\mathcal{F}$ we construct an algorithm A that takes a vector $(X, \beta_1, \ldots, \beta_Q; \rho)$, where $X \in \mathbb{G}^*$ is a DLP instance, $\beta_1, \ldots, \beta_Q \in \mathbb{B}$, and ρ is a set of random coins, as specified in Figure 8. The algorithm invokes $\mathcal{F}$ as a subroutine, on input X and a random tape derived from ρ, and processes bijective random oracle queries as they are processed in G_3. More precisely, queries to BRO are answered using the $\beta_1, \ldots, \beta_Q$ values, and queries to BRO^{-1} are answered using randomness derived from ρ. When $\mathcal{F}$ stops and outputs a forgery $(m^*, \langle s^*, t^*\rangle)$, algorithm A finds the index j of the BRO query corresponding to the forgery (which necessarily exists), and outputs it together with the triple (m^*, s^*, t^*). If $\mathcal{F}$ aborts also A aborts. In Section 2.4 the probability $\mathsf{acc} = \Pr[X \leftarrow_{\$} \mathbb{G}^*; \beta_1, \ldots, \beta_Q \leftarrow_{\$} \mathbb{B}; \mathsf{A}(X, \beta_1, \ldots, \beta_Q) \not\Rightarrow \perp]$ that A does not abort is referred to as its accepting probability. A comparison of A from Figure 8 with G_3 from Figure 7 shows that we have $\mathsf{acc} = \Pr[G_3^{\mathcal{F}} \Rightarrow 1]$.

Consider next the DLP solver $\mathcal{I}$ specified in Figure 9. It starts with invoking the forking algorithm $\mathsf{Fork}_{\mathsf{A}}$ (cf. Figure 2) as a subroutine (lines 01–09), on input X. The forking lemma (cf. Section 2.4) establishes that the probability frk that $\mathsf{Fork}_{\mathsf{A}}(X)$ terminates without aborting, i.e., that the execution of $\mathcal{I}$ reaches at least line 10, is bounded as $\mathsf{frk} \geq \mathsf{acc}(\mathsf{acc}/Q - 2^{-L})$.

To assess the success probability of $\mathcal{I}$ of solving DLP challenges it remains to bound the probability of abort in line 10, and to assert the effectiveness of the recovery step in line 11.

Concerning line 10, the probability of an abort can be bounded using the ψ-relative division advantage $\varepsilon_{\psi\mathsf{dr}}$ (cf. Definition 5) of a stateful adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ as in Figure 10. Concretely, $\mathcal{B}_1$ gets input β and guesses $j^* \in [1\,..\,Q]$. It then simulates lines 01 to 04 of $\mathcal{I}$ with the only difference that it overwrites $\beta_{j^*} \leftarrow \beta$. Once it gets $(j, (m, s, t))$ from A it checks if it guessed j correctly and forwards m to its own challenger, together with state $\Gamma = (X, j^*, \beta_1, \ldots, \beta_{j^*}, \rho)$. When $\mathcal{B}_2$ is invoked on input (Γ, β'), it simulates lines 05–09 of $\mathcal{I}$, overwriting $\beta'_{j^*} \leftarrow \beta'$. Once it gets m' from A, it forwards it to its challenger and wins if $H(m)/t = H(m')/t'$

Procedure INIT
00 $(\Pi_{\rightarrow}, \Pi_{\leftarrow}) \leftarrow (\emptyset, \emptyset)$
01 Obtain $(X, \beta_1, \ldots, \beta_Q; \rho)$
02 Return X

Procedure $\mathrm{BRO}(\alpha)$
as in G_3 (cf. Figure 7)

Procedure $\mathrm{BRO}^{-1}(\beta)$
as in G_3 (cf. Figure 7)

Procedure $\mathrm{FIN}(m^*, \langle s^*, t^* \rangle)$
03 If $s^* = 0 \vee t^* = 0$: Abort
04 $U^* \leftarrow g^{H(m^*)} X^{t^*}$
05 If $U^* = 1$: Abort
06 $R^* \leftarrow (U^*)^{1/s^*}$
07 $\alpha^* \leftarrow \varphi(R^*)$
08 $\beta^* \leftarrow \Pi(\alpha^*)$
09 // was queried before
10 If $\psi(\beta^*) \neq t^*$: Abort
11 If $(\alpha^*, \beta^*) \in \Pi_{\leftarrow}$: Abort
12 Find $j \in [1\,..\,k]$ s.t. $\beta^* = \beta_j$
13 Output j and (m^*, s^*, t^*)

Figure 8: Algorithm A is constructed from key-only forger $\mathcal{F}$. It obtains its input in line 01 and declares its output, unless it aborts, in line 13. Procedures $\mathrm{INIT}, \mathrm{BRO}, \mathrm{BRO}^{-1}, \mathrm{FIN}$ are invoked by $\mathcal{F}$. Note that BRO^{-1} contains instructions for sampling from sets $\mathbb{A}$ and $\mathbb{Z}_q^*$; in the context of A, these operations are understood to take their randomness from ρ. Also the randomness of $\mathcal{F}$ is taken from ρ. We write 'Abort' as shortcut for 'Output $\perp$ and stop'.

$\mathcal{I}(X)$
00 Run forking algorithm $\mathsf{Fork}_{\mathsf{A}}(X)$ as a subroutine:
01 Pick random coins ρ for A
02 $\beta_1, \ldots, \beta_Q \leftarrow_{\$} \mathbb{B}$
03 $(j, (m, s, t)) \leftarrow \mathsf{A}(X, \beta_1, \ldots, \beta_Q; \rho)$
04 // If A aborts outputting $\perp$: Abort
05 $\beta'_j, \ldots, \beta'_Q \leftarrow_{\$} \mathbb{B}$
06 Abort if $\beta_j = \beta'_j$
07 $(j', (m', s', t')) \leftarrow \mathsf{A}(X, \beta_1, \ldots, \beta_{j-1}, \beta'_j, \ldots, \beta'_Q; \rho)$
08 // If A aborts outputting $\perp$: Abort
09 Abort if $j \neq j'$
10 Abort if $H(m)/t = H(m')/t'$
11 Compute x as described in text
12 Output x

Figure 9: DLP-solver $\mathcal{I}$ runs algorithm A (twice) as a subroutine. We write 'Abort' as shortcut for 'Output $\perp$ and stop'.

(since $t = \psi(\beta_{j^*})$, $t' = \psi(\beta'_{j^*})$ and $t, t' \neq 0$). It follows that the probability of an abort in line 10 of $\mathcal{I}$ is bounded by $Q\varepsilon_{\psi\mathsf{dr}}$.

Now consider the case line 11 of Figure 9 is reached. Note that the same $\varphi(R)$ is used in both forgeries by construction of A and $\mathsf{Fork}_{\mathsf{A}}$, since the executions of $\mathcal{I}$ only differ from the jth BRO query onwards, so by the semi-injectiveness of φ we have $R^{\eta} = (g^{H(m)} X^t)^{\eta/s} = (g^{H(m')} X^{t'})^{1/s'} = R'$ for $\eta \in \{\pm 1\}$. Let $r = \log R, r' = \log R'$, and $x = \log X$. Then $r' = \eta r$ and hence $sr - tx = H(m)$ and $s'\eta r - t'x = H(m')$. By dividing the first equation by t and the second by t' and subtracting them from each other, it follows that $r(s/t - s'\eta/t') = H(m)/t - H(m')/t'$. Since the right-hand side is nonzero by the condition in line 10 and r is nonzero by validity of the forgery, so is $s/t - s'\eta/t'$ and $\mathcal{I}$ can compute $r = (H(m)/t - H(m')/t')/(s/t - s'\eta/t')$ and thus also $x = (sr - H(m))/t$.

Recall that adversary $\mathcal{F}$ (τ, Q, ε)-breaks the key-only security of GenDSA, that adversary $\mathcal{B}$ $(\tau_{\psi\mathsf{dr}}, \varepsilon_{\psi\mathsf{dr}})$-breaks the ψ-relative division resistance of H, and that there is an adversary that (τ'', ε'')-breaks the DLP in $\mathbb{G}$. As we saw, $\mathsf{acc} = \Pr[G_3^{\mathcal{F}} \Rightarrow 1] \geq \varepsilon - \varepsilon'' - Q^2/2^L$, and $\mathsf{frk} \geq \mathsf{acc}(\mathsf{acc}/Q - 1/2^L)$.

$\mathcal{B}_1(\beta)$
00 $x \leftarrow_{\$} \mathbb{Z}_q^*$; $X \leftarrow g^x$
01 $j^* \leftarrow_{\$} [1\,..\,Q]$
02 Pick random coins ρ for A
03 $\beta_1, \ldots, \beta_Q \leftarrow_{\$} \mathbb{B}$
04 $\beta_{j^*} \leftarrow \beta$
05 $L \leftarrow (\beta_1, \beta_2, \ldots, \beta_Q)$
06 $(j, (m, s, t)) \leftarrow \mathsf{A}(X, L; \rho)$
07 // If A outputs $\perp$: Abort
08 Abort if $j \neq j^*$
09 $\Gamma \leftarrow (X, j^*, \beta_1, \ldots, \beta_{j^*}, \rho)$
10 Return (m, Γ)

$\mathcal{B}_2(\Gamma, \beta')$
11 Recover $X, j^*, \beta_1, \ldots, \beta_{j^*}, \rho$
12 $\beta'_{j^*}, \ldots, \beta'_Q \leftarrow_{\$} \mathbb{B}$
13 $\beta'_{j^*} \leftarrow \beta'$
14 Abort if $\beta_{j^*} = \beta'_{j^*}$
15 $L' \leftarrow (\beta_1 \ldots \beta_{j^*-1}, \beta'_{j^*} \ldots \beta'_Q)$
16 $(j', (m', s', t')) \leftarrow \mathsf{A}(X, L'; \rho)$
17 // If A outputs $\perp$: Abort
18 Abort if $j \neq j'$
19 Return m'

Figure 10: Construction of adversary $\mathcal{B}$ against the ψ-relative division resistance of H. We write 'Abort' as a shortcut for 'Return $\perp$'.

We can assume w.l.o.g. that $1/2^L \leq \mathsf{acc}/2Q$, because otherwise ε can be bounded by $\varepsilon'' + Q(Q+2)/2^L$ and we are done. Now overall, for the success probability ε' of $\mathcal{I}$ we have $\varepsilon' \geq \mathsf{frk} - Q\varepsilon_{\psi\mathsf{dr}} \geq \mathsf{acc}(\mathsf{acc}/Q - 1/2^L) - Q\varepsilon_{\psi\mathsf{dr}} \geq \mathsf{acc}(\mathsf{acc}/Q - \mathsf{acc}/2Q) - Q\varepsilon_{\psi\mathsf{dr}} = \mathsf{acc}^2/2Q - Q\varepsilon_{\psi\mathsf{dr}} \geq (\varepsilon - \varepsilon'' - Q^2/2^L)^2/2Q - Q\varepsilon_{\psi\mathsf{dr}}$. All in all we proved $\Pr[\mathrm{KO}^{\mathcal{F}} \Rightarrow 1] \leq \sqrt{2Q^2\varepsilon_{\psi\mathsf{dr}} + 2Q\varepsilon'} + \varepsilon'' + Q^2/2^L$, as required. □

5. CONCLUSIONS FOR DSA AND ECDSA

By combining Theorems 1 and 2 we obtain that if φ is semi-injective and H is sufficiently secure then GenDSA is euf-cma secure if the discrete logarithm problem in $\mathbb{G}$ is hard, in the bijective random oracle model for Π. Further, in the full version we show that GenDSA is strongly unforgeable if and only if φ is injective. In practice, the conditions on H and $\mathbb{G}$ are arguably fulfilled for the standardized versions of DSA and ECDSA. That is, if one accepts the random bijection model for Π, our overall results in a nutshell are: DSA signatures are strongly unforgeable, and ECDSA signatures are existentially unforgeable.

6. ACKNOWLEDGMENTS

Supported by ERC Project ERCC (FP7/615074).

7. REFERENCES

[1] M. Bellare, G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In A. Juels, R. N. Wright, S. Vimercati, eds., *ACM CCS 06*: 390-399, Alexandria, US, Oct. 2006. ACM Press.

[2] E. Brickell, D. Pointcheval, S. Vaudenay, M. Yung. Design validations for discrete logarithm based signature schemes. In H. Imai, Y. Zheng, Eds., *PKC 2000*, *LNCS 1751*: 276-292, Melbourne, AU, Jan. 2000. Springer.

[3] D. Brown. On the provable security of ECDSA. In I. Blake, G. Seroussi, N. Smart, Eds., *Advances in Elliptic Curve Cryptography*: 21-40. Cambr. Uni. Pr., 2005

[4] D. Brown. Generic groups, collision resistance, and ECDSA. IACR ePrint 2002/026. http://ia.cr/2002/026.

[5] D. Brown. Generic groups, collision resistance, and ECDSA. *Des. Codes Cryptography*, 35(1):119-152, 2005.

[6] D. Brown. One-up problem for (EC)DSA. IACR ePrint 2008/286. http://ia.cr/2008/286.
[7] A. W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Y. Zheng, Ed., *ASIACRYPT 2002*, *LNCS 2501*: 100-109, Queenstown, NZ, Dec. 2002. Springer.
[8] V. Dolmatov, A. Degtyarev. GOST R 34.10-2012: Digital Signature Algorithm. RFC 7091, Dec. 2013. http://www.ietf.org/rfc/rfc7091.txt.
[9] N. Howgrave-Graham, N. Smart. Lattice attacks on digital signature schemes. *Des. Codes Crypt.*, 23(3):283-290, 2001.
[10] ISO/IEC 11889:2015. Information technology - trusted platform module library, 2013.
[11] C. Kerry, P. Gallagher. FIPS PUB 186-4: Digital Signature Standard (DSS), 2013. http://dx.doi.org/10.6028/NIST.FIPS.186-4.
[12] E. Kiltz, K. Pietrzak. Leakage resilient ElGamal encryption. In M. Abe, Ed., *ASIACRYPT 2010*, *LNCS 6477*: 595-612, SG, Dec. 2010. Springer.
[13] P. J. Leadbitter, D. Page, N. P. Smart. Attacking DSA under a repeated bits assumption. In M. Joye, J.-J. Quisquater, Eds., *CHES 2004*, *LNCS 3156*: 428-440, Cambridge, US, Aug. 2004. Springer.
[14] J. Malone-Lee, N. P. Smart. Modifications of ECDSA. In K. Nyberg, H. M. Heys, Eds., *SAC 2002*, *LNCS 2595*: 1-12, St. John's, CA, Aug. 2003. Springer.
[15] U. Maurer. Abstract models of computation in cryptography. In N. P. Smart, Ed., *10th IMA International Conference on Cryptography and Coding*, *LNCS 3796*: 1-12, Cirencester, UK, Dec. 2005. Springer.
[16] U. Maurer, S. Wolf. Lower bounds on generic algorithms in groups. In K. Nyberg, Ed., *EUROCRYPT'98*, *LNCS 1403*: 72-84, Espoo, FI, May/June 1998. Springer.
[17] G. Neven, N. P. Smart, B. Warinschi. Hash function requirements for Schnorr signatures. *J. Mathematical Cryptology*, 3(1):69-87, 2009.
[18] P. Q. Nguyen, I. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Des. Codes Crypt.*, 30(2):201-217, 2003.
[19] P1363. IEEE P1363-2000: Standard specifications for public key cryptography. IEEE, Inc., 2000. http://grouper.ieee.org/groups/1363/.
[20] D. Pointcheval, J. Stern. Security proofs for signature schemes. In U. Maurer, Ed., *EUROCRYPT'96*, *LNCS 1070*: 387-398, Saragossa, ES, May 1996. Springer.
[21] D. Pointcheval, S. Vaudenay. On provable security for digital signature algorithms. Technical report, Technical Report LIENS-96-17, LIENS, 1996.
[22] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, Ed., *EUROCRYPT'97*, *LNCS 1233*: 256-266, Konstanz, DE, May '97. Springer.
[23] N. P. Smart. The exact security of ECIES in the generic group model. In B. Honary, Ed., *8th IMA International Conference on Cryptography and Coding*, *LNCS 2260*: 73-84, Cirencester, UK, Dec. 2001. Springer.
[24] J. Stern, D. Pointcheval, J. Malone-Lee, N. P. Smart. Flaws in applying proof methodologies to signature schemes. In M. Yung, Ed., *CRYPTO 2002*, *LNCS 2442*: 93-110, Sta. Barbara, US, Aug. 2002. Springer.
[25] S. Vaudenay. Hidden collisions on DSS. In N. Koblitz, Ed., *CRYPTO'96*, *LNCS 1109*: 83-88, Sta. Barbara, US, Aug. 1996. Springer.
[26] S. Vaudenay. The security of DSA and ECDSA. In Y. Desmedt, Ed., *PKC 2003*, *LNCS 2567*: 309-323, Miami, US, Jan. 2003. Springer.
[27] Z. Zhang, K. Yang, J. Zhang, C. Chen. Security of the SM2 signature scheme against generalized key substitution attacks. In L. Chen, S. Matsuo, Eds., *Security Standardisation Research 2015*, *LNCS 9497*: 140-153, Tokyo, JP, Dec. 2015. Springer.

APPENDIX

A. FUNCTIONS IN THE GGM

We consider how the generic group model should be used for the analysis of DL-based signature schemes that make use of a non-algebraic component such as a hash function. Although they have never been written down at a formal level (to the best of our knowledge), most of the observations made in this section can be considered folklore. We thank Ueli Maurer and others for useful discussions in this context.

Shoup [22] introduced the generic group model for studying cryptographic problems ('hardness assumptions') related to the discrete logarithm problem in the context of *generic algorithms*. That is, "algorithms that do not exploit any special properties of the encodings of group elements other than the property that each group element is encoded as a unique binary string" [22]. Generic group models proved useful for establishing generic lower bounds for several newly proposed cryptographic problems. Even though no concrete group can be fully generic, it is generally believed in cryptography that the generic group model approximates the behavior of elliptic curve groups 'well-enough' in most cases.

The generic group model for algebraic problems.

Let $\Pi[\mathbb{G}]$ be a *cryptographic problem* defined over some prime-order group $(\mathbb{G}, g, q)$ which is expressed via a (potentially interactive) probabilistic experiment between a challenger and an unbounded adversary. Suppose $\Pi[\mathbb{G}]$ is *algebraic* in the sense that all operations of the experiment can be described by group operations only. An example for an algebraic cryptographic problem is the DLP; in contrast, a cryptographic problem involving a hash function $f\colon \mathbb{G} \to \{0,1\}^k$ (e.g., providing collision resistance) is not algebraic. In order to analyze problem $\Pi[\mathbb{G}]$ in the generic group model, one commonly executes the following experiment. (There exist variants of the generic group model, but the choice of the variant does not affect our arguments.)

In the beginning, a random bijective encoding function

$$E\colon \mathbb{G} \to \mathbb{A}_q := \{0, \ldots, q-1\} \tag{4}$$

is chosen (from the set of all possible such functions), where $\mathbb{A}_q$ is naturally interpreted as a subset of $\{0,1\}^m$ with $m = \lceil \log(q) \rceil$. Set $\mathbb{A}_q$ can be viewed as an abstract copy of $\mathbb{G}$, but itself without any algebraic structure.

The interactive security experiment $\Pi[\mathbb{G}]$ is then executed with the adversary where group $\mathbb{G}$ is substituted by $\mathbb{A}_q$,

i.e., every appearance of a group element $A \in \mathbb{G}$ is replaced by the encoding $E(A)$ of it. More precisely, if the experiment is supposed to send $A \in \mathbb{G}$ to the adversary, it sends $A' := E(A) \in \mathbb{A}_q$ instead; if the adversary is supposed to send $A \in \mathbb{G}$ to the experiment, it sends some value $A' \in \mathbb{A}_q$ instead which is interpreted by the experiment as $A = E^{-1}(A')$. All other messages, including elements from $\mathbb{Z}_q$, are exchanged unmodified. Importantly, the adversary only sees encodings of group elements of the form $E(A)$ but it gets neither access to E itself nor to any group element held by the experiment. Rather, it performs group operations through the group oracle

$$\mathsf{mult}\colon \mathbb{A}_q \times \mathbb{A}_q \to \mathbb{A}_q;\ (A', B') \mapsto E(E^{-1}(A')E^{-1}(B'))\ .$$

Note $(\mathbb{A}_q, \mathsf{mult})$ can be considered a 'group' with neutral element $1' = E(1)$.

The intuition behind the described framework is that it models an adversary that cannot exploit any specific property of the underlying group $\mathbb{G}$. This is a clean and well-accepted model. It allows, for example, to prove that any otherwise unbounded generic adversary that makes at most t queries to the group oracle can solve the discrete logarithm problem with probability at most $O(t^2/q)$ [22].

Generic groups for non-algebraic problems.

Once we drop the requirement that the cryptographic problem be algebraic it becomes unclear how to adapt the generic group model in order to obtain meaningful results. Suppose the cryptographic problem $\Pi[\mathbb{G}, f]$ involves hash function $f\colon \mathbb{G} \to \{0,1\}^k$ that takes group elements and outputs bit-strings such that the behavior of the function cannot be described by group operations only (we call this a non-algebraic function). In fact, in such problems, function f really operates on bit-strings of some length n and one silently assumes an encoding that represents each group element $A \in \mathbb{G}$ as a unique bit-string of this length. It is thus more precise to say that hash function f describes how it acts on group elements that are represented by binary strings. For example, if $\mathbb{G}$ is an elliptic curve group over $\mathrm{GF}(p)$, one could set $n = 2\lceil \log(p) \rceil$ and represent group elements $A = (x, y) \in \mathrm{GF}(p) \times \mathrm{GF}(p)$ in the canonic form; further, $f\colon \{0,1\}^n \to \{0,1\}^k$ could be a member of the SHA family.

How should one consider such a non-algebraic function f in the generic group model? This turns out to be rather unclear. The issue is that the generic group adversary only sees elements from $\mathbb{A}_q$, not from $\mathbb{G}$, and yet, intuitively, any reasonable model has to allow the adversary the evaluation of f on group elements of its choice. Note here that one cannot simply provide the adversary with oracle access to f via $\mathcal{O}_f(A') := f(E^{-1}(A'))$ since $\mathcal{O}_f(A')$ may leak information about group element $A = E^{-1}(A') \in \mathbb{G}$ (as this information could be exploited by the computationally unbounded adversary, this would contradict the idea of the GGM). So in order to give the adversary fair access to function f we have to define another function $f'\colon \mathbb{A}_q \to \{0,1\}^k$ that somehow mimics the behavior of $f\colon \mathbb{G} \to \{0,1\}^k$. The following approach to define f' was taken in previous generic group proofs (e.g., [23, 5, 17, 12]). There, $f\colon \mathbb{G} \to \{0,1\}^k$ is simply transferred, without any modification, to $\mathbb{A}_q$, i.e., the generic group proof is carried out with respect to function

$$f'\colon \mathbb{A}_q \to \{0,1\}^k, \quad \text{where } f'(A') := f(A')\ , \qquad (5)$$

instead of f. Note, however, that $\mathbb{A}_q$ and $\mathbb{G}$ are independent, i.e., setting $f'(A') := f(A')$ is generally not even well-defined! But as $\mathbb{A}_q \subseteq \{0,1\}^m$ and $\mathbb{G} \subseteq \{0,1\}^n$, they both act on bit-strings and can usually be made well-defined by forcing $\mathbb{A}_q \subseteq \mathbb{G}$. Now, if we successfully carried out a generic group proof with respect to function f' defined in (5), for which concrete function $f''\colon \mathbb{G} \to \{0,1\}^k$ did we actually prove the generic hardness of Π? Given $f'\colon \mathbb{A}_q \to \{0,1\}^k$, we can express $f''\colon \mathbb{G} \to \{0,1\}^k$ via the random bijection E from (4) as $f'' = f' \circ E = f \circ E$. Hence a generic group proof using the approach to replace function f by f' implicitly proves the hardness of $\Pi[\mathbb{G}, f'']$. Note, however, that $f'' = f \circ E$ is necessarily an *idealized object* (as E is one). It thus remains unclear how much impact a security result on $\Pi[\mathbb{G}, f'']$ has on the original problem $\Pi[\mathbb{G}, f]$. In particular, the generic group proof can now make use of the fact that each group element is encoded as a unique *random* binary string, not only a unique binary string as motivated by Shoup [22].

Maurer's generic model for non-algebraic problems.

Maurer's generic model of computation [15] differs from the one by Shoup in that the adversary never obtains representations $E(A)$ of group elements A, but can only access known group elements via abstract handles. Consequently, in this model it is unavoidable that any non-algebraic function $f\colon \mathbb{G} \to \{0,1\}^k$ is modeled via explicit oracle access. This solves the main problem of non-algebraic functions in the generic group model we encountered above, namely that it is not clear how the adversary can access function f. For this reason we believe that Maurer's model offers a much cleaner and rigorous way to analyze non-algebraic problems in a generic model.

The security of ECDSA in the generic group model.

Brown provides a security proof of ECDSA in the generic group model [5, 3], assuming $f' := f$ as in (5), where f is the 'mod q' conversion function of ECDSA. By our discussion we can conclude that his proof implicitly considers an idealized conversion function, namely ECDSA with conversion function $f'' = f \circ E$, which is some idealized version of the real 'mod q' conversion function f.

In order to obtain a rigorous analysis of the generic security of ECDSA, one can try to translate Brown's proof to Maurer's generic model of computation. However, it remains unclear how to setup the oracle for the conversion function. A canonical candidate would be to give access to $f \circ E$ as discussed above, but the problem is that Brown's proofs require the conversion function to be almost-invertible, which is arguably wrong for this function. We suspect that our techniques can be used to restate and prove Brown's result in Maurer's model of computation by defining $f = \psi \circ \Pi \circ \phi$ and providing the generic adversary oracle access to a bijective random oracle $\mathcal{O}_\Pi$. However, we did not further look into this, as our results imply stronger and more general security guarantees for ECDSA. We stress that there is no reason to believe that Brown's proofs are flawed. But, as discussed above, their interpretation and impact to ECDSA remain unclear.