A preliminary version of this paper appears in the Public-Key Cryptography (PKC) 2023 conference. This is the full version.

Hardening Signature Schemes via Derive-then-Derandomize: Stronger Security Proofs for EdDSA

Mihir Bellare[1] Hannah Davis[2] Zijing Di[3]

February 2023

Abstract

We consider a transform, called Derive-then-Derandomize, that hardens a given signature scheme against randomness failure and implementation error. We prove that it works. We then give a general lemma showing indifferentiability of **Shrink-MD**, a class of constructions that apply a shrinking output transform to an MD-style hash function. Armed with these tools, we give new proofs for the widely standardized and used EdDSA signature scheme, improving prior work in two ways: (1) we give proofs for the case that the hash function is an MD-style one, reflecting the use of SHA512 in the NIST standard, and (2) we improve the tightness of the reduction so that one has guarantees for group sizes in actual use.

[1] Department of Computer Science & Engineering, University of California, San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: `mbellare@ucsd.edu`. URL: `http://cseweb.ucsd.edu/~mihir/`. Supported in part by NSF grant CNS-2154272.

[2] Department of Computer Science & Engineering, University of California, San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: `h3davis@ucsd.edu`. URL: `http://cseweb.ucsd.edu/~h3davis/`. Supported in part by NSF grant CNS-2154272.

[3] Stanford University. `zidi@stanford.edu`.

Contents

1 Introduction 3

2 Preliminaries 7

3 Functor framework 8

4 The soundness of Derive-then-Derandomize 10

5 Security of EdDSA 14

6 Indifferentiability of the shrink-MD class of functors 20

References 28

A The unique order-p subgroup of $\mathbb{G}$ 30

B Group Instantiation for Ed25519 31

1 Introduction

In designing schemes, and proving them secure, theoreticians implicitly assume certain things, such as on-demand fresh randomness and correct implementation. In practice, these assumptions can fail. Weaknesses in system random-number generators are common and have catastrophic consequences. (An example relevant to this paper is the well-known key-recovery attack on $\mathsf{Schnorr}$ signatures when signing reuses randomness. Another striking example are Ps and Qs attacks [24, 28].) Meanwhile, implementation errors can be exploited, as shown by Bleichenbacher's attack on RSA signatures [14].

In light of this, practitioners may try to "harden" theoretical schemes before standardization and usage. A prominent and highly successful instance is EdDSA, a hardening of the $\mathsf{Schnorr}$ signature scheme proposed by Bernstein, Duif, Lange, Schwabe, and Yang (BDLSY) [13]. It incorporates explicit, simple key-derivation, makes signing deterministic, adds protection against sidechannel attacks via "clamping," and for simplicity confines itself to a single hash function, namely $\mathtt{SHA512}$. The scheme is widely standardized [33, 26] and used [25].

There is however a subtle danger here, namely that the hardening attempt introduces new vulnerabilities. In other words, hardening needs to be done right; if not, it may even "soften" the scheme! Thus it is crucial that the hardened scheme be vetted via a proof of security. This is of particular importance for EdDSA given its widespread deployment. In that regard, Brendel, Cremers, Jackson and Zhao (BCJZ) [15] showed that EdDSA is secure if the Discrete-Log (DL) problem is hard and the hash function is modeled as a random oracle. This is significant as a first step but has at least two important limitations: (1) Due to the extension attack, a random oracle is not an appropriate model for the $\mathtt{SHA512}$ hash function EdDSA actually uses, and (2) the reduction is so loose that there is no security guarantee for group sizes in use today.

Extrapolating EdDSA, the first part of this paper defines a general hardening transform on signature schemes called Derive-then-Derandomize (**DtD**), and proves its soundness. Next we prove the indifferentiability of a general class of constructions, that we call shrink-MD; it includes the well-studied chop-MD construction [18] and also the modulo-a-prime construction arising in EdDSA. Armed with these results, the second part of the paper returns to give new proofs for EdDSA that in particular fill the above gaps. We begin with some background.

Respecting hash structure in proofs. Recall that the MD-transform [30, 19] defines a hash function $\mathsf{H} = \mathbf{MD}[\mathsf{h}] : \{0,1\}^* \to \{0,1\}^{2k}$ by iterating an underlying compression function $\mathsf{h} : \{0,1\}^{b+2k} \to \{0,1\}^{2k}$. (See Section 2 for details.) $\mathtt{SHA256}$ and $\mathtt{SHA512}$ are obtained in this way, with (b, k) being $(512, 128)$ and $(1024, 256)$, respectively. This structure gives rise to attacks, of which the most well known is the extension attack. The latter allows an attacker given $t \leftarrow \mathbf{MD}[\mathsf{h}](e_2 \| M)$, where e_2 is a secret unknown to the attacker and $M \in \{0,1\}^*$ is public, to compute compute $t' = \mathbf{MD}[\mathsf{h}](e_2 \| M')$, for some $M' \in \{0,1\}^*$ of its choice. This has been exploited to violate the UF-security of the so-called prefix message authentication code $\mathsf{pfMAC}_{e_2}(M) = \mathsf{H}(e_2 \| M)$ when H is an MD-hash function; HMAC [4] was designed to overcome this.

A proof of security of a scheme (such as EdDSA) that uses a hash function H will often model H as a random oracle [9], in what we'll call the (H, H)-model: scheme algorithms, and the adversary, both have oracle access to the same random H. However the presence of the above-discussed structure in "real" hash functions led Dodis, Ristenpart and Shrimpton (DRS) [20] to argue that the "right" model in which to prove security of a scheme that uses $\mathsf{H} = \mathbf{MD}[\mathsf{h}]$ is to model the compression function h —rather than the hash function $\mathsf{H} = \mathbf{MD}[\mathsf{h}]$— as a random oracle. We'll call this the $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$-model: the adversary has oracle access to a random h, with scheme algorithms having access to $\mathbf{MD}[\mathsf{h}]$. There is now widespread agreement with the DRS thesis that proofs of security of MD-hash-using schemes should use the $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$ model.

Giving from-scratch proofs in the $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$ model is, however, difficult. Maurer, Renner and Holenstein (MRH) [29] show that if a construction $\mathbf{F}$ is indifferentiable (abbreviated indiff) and a scheme is secure in the (H, H) model, then it remains secure in the $(\mathbf{F}[\mathsf{h}], \mathsf{h})$ model. (This requires the game defining security of the scheme to be single-stage [37], which is true for the relevant ones here.) Unfortunately, $\mathbf{F} = \mathbf{MD}$ is provably *not* indiff [18], due exactly to the extension attack. So the MRH result does not help with $\mathbf{MD}$. This led to a search for indiff variants. DRS [20] and YMO [41] (independently) offer public-indiff and show that it suffices to prove security, in the $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$ model, of schemes that use $\mathbf{MD}$ in some restricted way. However, EdDSA does not obey these restrictions. Thus, other means are needed.

THE EdDSA SCHEME. The Edwards curve Digital Signature Algorithm (EdDSA) is a Schnorr-based signature scheme introduced by Bernstein, Duif, Lange, Schwabe and Yang [13]. Ed25519, which uses the Curve25519 Edwards curve and SHA512 as the hash function, is its most popular instance. The scheme is standardized by NIST [33] and the IETF [26]. It is used in TLS 1.3, OpenSSH, OpenSSL, Tor, GnuPGP, Signal and WhatsApp. It is also the preferred signature scheme of the Corda, Tezos, Stellar and Libra blockchain systems. Overall, IANIX [25] reports over 200 uses of Ed25519. Proving security of this scheme is accordingly of high importance.

Figure 4 shows EdDSA on the right, and, on the left, the classic Schnorr scheme [39] on which EdDSA is based. The schemes are over a cyclic, additively-written group $\mathbb{G}$ of prime order p with generator B. The public verification key is A. The Schnorr hash function has range $\mathbb{Z}_{\mathsf{p}} = \{0, \ldots, \mathsf{p}-1\}$, while, for EdDSA, function H_1 has range $\{0,1\}^{2k}$ where k, the bit-length of p, is 256 for Ed25519. Functions $\mathsf{H}_2, \mathsf{H}_3$ have range $\mathbb{Z}_{\mathsf{p}}$.

EdDSA differs from Schnorr in significant ways. While the Schnorr secret key s is in $\mathbb{Z}_{\mathsf{p}}$, the EdDSA secret key sk is a k-bit string. This is hashed and the $2k$-bit result is split into k-bit halves $e_1 \| e_2$. A Schnorr secret-key s is derived by applying to e_1 a clamping function CF that zeroes out the three least significant bits of e_1. (Note: This means s is *not* uniformly distributed over $\mathbb{Z}_{\mathsf{p}}$.) Clamping increases resistance to side-channel attacks [13]. Signing is made deterministic by a standard de-randomization technique [22, 32, 8, 11], namely obtaining the Schnorr randomness r by hashing the message M with a secret-key dependent string e_2. We note that all of $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$ are instantiated via the same hash function, namely SHA512.

PRIOR WORK AND OUR QUESTIONS. Recall that the security goal for a signature scheme is UF (UnForgeability under Chosen-Message Attack) [23]. Schnorr is well studied, and proven UF under DL (Discrete Log in $\mathbb{G}$) when H is a random oracle [36, 1]. The provable security of EdDSA, however, received surprisingly little attention until the work of Brendel, Cremers, Jackson and Zhao (BCJZ) [15]. They take the path also used for Schnorr and other identification-based signature schemes [36, 1], seeing EdDSA as the result of the Fiat-Shamir transform on an underlying identification scheme EdID that they define, proving security of the latter under DL, and concluding UF of EdDSA under DL when H is a random oracle. This is an important step forward, but the BCJZ proof [15] remains in the (H, H) model. We ask and address the following two questions.

1. Can we prove security in the $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$ model? The NIST standard [33] mandates that Ed25519 uses SHA512, which is an MD-hash function. Accordingly, as explained above, the BCJZ proof [15], being in the (H, H) model, does not guarantee security; to do the latter, we need a proof in the $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$ model.

The gap is more than cosmetic. As we saw above with the example of the prefix MAC, a scheme could be secure in the (H, H) model, yet totally insecure in the more realistic $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$ model, and thus also in practice. And EdDSA skirts close to the edge: line 14 is using the prefix-MAC that the extension attack breaks, and overlaps in inputs across the three uses of H could lead to failures.

Intuitively what prevents attacks is that the MAC outputs are taken modulo p, and inputs to H in two of the three uses involve secrets. Thus, we'd expect that the scheme is indeed secure in the $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$ model.

Proving this, however, is another matter. We already know that **MD** is not indiff. It is public indiff [20, 41], but this will not suffice for EdDSA because $\mathsf{H}_1, \mathsf{H}_2$ are being called on secrets. We ask, first, can EdDSA be proved secure in the $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$ model, and second, can this be done in some modular way, rather than from scratch?

2. Can we improve reduction tightness? The reduction of BCJZ [15] is so loose that, in the 256-bit curve over which Ed25519 is implemented, it guarantees little security. Let's elaborate. Given an adversary A_{UF} violating the UF-security of EdDSA with probability ϵ_{UF}, the reduction builds an adversary A_{DL} breaking DL with probability $\epsilon_{\mathrm{DL}} = \epsilon_{\mathrm{UF}}^2/q_h$ where q_h is the number of H-queries of A_{UF} and the two adversaries have about the same running time t. (The square arises from the use of rewinding, analyzed via the Reset Lemma of [7].) In an order p elliptic curve group, $\epsilon_{\mathrm{DL}} \approx t^2/p$ so we get $\epsilon_{\mathrm{UF}} = t \cdot \sqrt{q_h/p}$. Ed25519 has $p \approx 2^{256}$. Say $t = q_h = 2^{70}$, which (as shown by BitCoin mining capability) is not far from attacker reach. Then $\epsilon_{\mathrm{DL}} = 2^{-116}$ is small but $\epsilon_{\mathrm{UF}} = 2^{70} \cdot 2^{-(256-70)/2} = 2^{-23}$ is in comparison quite high.

Now, one might say that one would not expect better because the same reduction loss is present for Schnorr. The classical reductions for Schnorr [36, 1] did indeed display the above loss, but that has changed: recent advances for Schnorr include a tighter reduction from DL [38], an almost-tight reduction from the MBDL problem [5] and a tight reduction from DL in the Algebraic Group Model [21]. We'd like to put EdDSA on par with the state of the art for Schnorr. We ask, first, is this possible, and second, is there a modular way to do it that leverages, rather than repeats, the (many, complex) just-cited proofs for Schnorr?

CONTRIBUTIONS FOR EdDSA. We simultaneously simplify and strengthen the security proofs for EdDSA as follows.

1. Reduction from Schnorr. Rather than, as in prior work, give a reduction from DL or some other algebraic problem, we give a simple, direct reduction from Schnorr itself. That is, we show that if the Schnorr signature scheme is UF-secure, then so is EdDSA. Furthermore, the reduction is *tight* up to a constant factor. This allows us to leverage prior work [38, 5, 21] to obtain tight proofs for EdDSA under various algebraic assumptions and justify security for group sizes in actual use. But there are two further dividends. First, Schnorr [39] is over 30 years old and has withstood the tests of time and cryptanalysis, so our proof that EdDSA is just as secure as Schnorr allows the former to inherit, and benefit from, this confidence. Second, our result formalizes and proves what was the intuition and belief in the first place [13], namely that, despite the algorithmic differences, EdDSA is a sound hardening of Schnorr.

2. Accurate modeling of the hash function. As noted above, BCJZ [15] assume the hash function H is a random oracle, but this, due to the extension attack, is not an accurate model for the MD-hash function SHA512 used by EdDSA. We fill this gap by instead proving security in the $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$ model, where $\mathsf{H} = \mathbf{MD}[\mathsf{h}]$ is derived via the MD-transform [30, 19] and the compression function h is a random oracle.

APPROACH AND BROADER CONTRIBUTIONS. The above-mentioned results on EdDSA are obtained as a consequence of more general ones.

3. The DtD transform and its soundness. We extend the hardening technique used in EdDSA to define a general transform that we call Derive-then-Derandomize (**DtD**). It takes an *arbitrary* signature scheme DS, and with the aid of a PRG H_1 and a PRF H_2, constructs a hardened signature scheme $\overline{\mathsf{DS}}$. We provide (Theorem 4.1) a strong and general validation of **DtD**, showing that $\overline{\mathsf{DS}}$ is UF-secure assuming DS is UF-secure. Moreover *the reduction is tight* and the proof is simple. This shows that the EdDSA hardening method is generically sound.

4. Indifferentiability of Shrink-MD. It is well-known that **MD** is not indifferentiable [29] from a random oracle, but that the **Chop-MD** [18], which truncates the output of an an **MD** hash by some number of bits, is indifferentiable. Unfortunately, we identified gaps in two prominent proofs of indifferentiability of **Chop-MD** [18, 31]. EdDSA uses a similar construction that reduces the **MD** hash output modulo a prime p sufficiently smaller than the size of the range of **MD**, due to which we refer to this construction as **Mod-MD**. The **Mod-MD** construction has not been proven indifferentiable. We simultaneously give new proofs of indifferentiability for **Chop-MD** and **Mod-MD** as part of a more general class of constructions that we call **Shrink-MD** functors. These are constructions of the form $\mathsf{Out}(\mathbf{MD})$ where Out is some output-processing function, and we prove indifferentiability under certain "shrinking" conditions on Out.

5. Application to EdDSA. EdDSA is obtained as the result $\overline{\mathsf{DS}}$ of the **DtD** transform applied to the $\mathsf{DS} = \mathsf{Schnorr}$ signature scheme, and with the PRG and PRF defined via **MD**, specifically $\mathsf{H}_1(sk) = \mathbf{MD}[\mathsf{h}](sk)$ and $\mathsf{H}_2(e_2, M) = \mathbf{MD}[\mathsf{h}](e_2 \| M) \bmod \mathsf{p}$ where p is the prime order of the underlying group. Additionally, the hash function used in Schnorr is also $\mathsf{H}_3(X) = \mathbf{MD}[\mathsf{h}](X) \bmod \mathsf{p}$. Due to Theorem 4.1 validating **DtD**, we are left to show the PRG security of H_1, the PRF security of H_2 and the UF-security of Schnorr, all with h modeled as a random oracle. We do the first directly. We obtain the second as a consequence of the indifferentiability of **Mod-MD**. (In principle it follows from the PRF security of AMAC [3], but we found it difficult to extract precise bounds via this route.) For the third, we again exploit indifferentiability of **Mod-MD**, together with a technique from BCJZ [15] to handle clamping, to reduce to the UF security of regular Schnorr, where the hash function is modeled as a random oracle. Putting all this carefully together yields our above-mentioned results for EdDSA. We note that one delicate and important point is that the idealized compression function h is *the same* across $\mathsf{H}_1, \mathsf{H}_2$ and H_3, meaning these are not independent. This is handled through the building blocks in Theorem 4.1 being functors [6] rather than functions.

Discussion and related work. Both BCJZ [15] and CGN [16] note that there are a few versions of EdDSA out there, the differences being in their verification algorithms. What Figure 4 shows is the most basic version of the scheme, but we will be able to cover the variants too, in a modular way, by reducing from Schnorr with the same verification algorithm.

BBT [3] define the function $\mathsf{AMAC}[\mathsf{h}]$ to take a key e_2 and message M, and return $\mathbf{MD}[\mathsf{h}](e_2 \| M) \bmod \mathsf{p}$. This is the H_2 in EdDSA. We could exploit their results to conclude PRF security of H_2, but it requires putting together many different pieces from their work, and it is easier and more direct to establish PRF security of H_2 by using our lemma on the indifferentiability of **Mod-MD**.

In the Generic Group Model (GGM) [40], it is possible to prove UF-security of Schnorr under standard (rather than random oracle) model assumptions on the hash functions [34, 17]. But use of the GGM means the result applies to a limited class of adversaries. Our results, following the classical proofs for identification-based signatures [36, 35, 1, 27], instead use the standard model for the group, while modeling the hash function (in our case, the compression function) as a random oracle.

In an earlier version of this paper, our proofs had relied on a variant of indifferentiability that we had introduced. At the suggestion of a Crypto 2022 reviewer, this has been dropped in favor of a direct proof based on PRG and PRF assumptions on $\mathsf{H}_1, \mathsf{H}_2$. We thank the (anonymous) reviewer for this suggestion.

Theorem 4.1 is in the standard model if the PRG, PRF and starting signature scheme DS are standard-model, hence can be viewed as a standard-model justification of the hardening template underlying EdDSA. However, when we want to justify EdDSA itself, we need to consider the specific, $\mathbf{MD}$-based instantiations of the PRG, PRF and $\mathsf{Schnorr}$ hash function, and for these we use the model where the compression function is ideal.

Several works study de-randomization of signing by deriving the coins via a PRF applied to the message, considering different ways to key the PRF [22, 32, 8, 11]. We use their techniques in the proof of Theorem 4.1.

One might ask how to view the UF-security of $\mathsf{Schnorr}$ signatures as an assumption. What is relevant is not its form (it is interactive) but that (1) it can be seen as a hub from where one can bridge to other assumptions that imply it, such as DL (non-tightly) [36, 1] or MBDL (tightly) [5], and (2) it is validated by decades of cryptanalysis.

Our results have been stated for UF but extend to SUF (Strong unforgeability), meaning our proofs also show SUF-security of EdDSA in the $(\mathbf{MD}[\mathsf{h}], \mathsf{h})$ model assuming SUF security of $\mathsf{Schnorr}$, with a tight (up to the usual constant factor) reduction.

EdDSA could be used with other hash functions such as $\mathtt{SHAKE}$. The extension attack does not apply to the latter, so the proof of BCJZ [15] applies, but gives a loose reduction from DL; our results still add something, namely a tight reduction from $\mathsf{Schnorr}$ and thus improved tightness in several ways as discussed above.

2 Preliminaries

NOTATION. If n is a positive integer, then $\mathbb{Z}_n$ denotes the set $\{0, \ldots, n-1\}$ and $[n]$ or $[1..n]$ denote the set $\{1, \ldots, n\}$. If $\boldsymbol{x}$ is a vector then $|\boldsymbol{x}|$ is its length (the number of its coordinates), $\boldsymbol{x}[i]$ is its i-th coordinate and $[\boldsymbol{x}] = \{\, \boldsymbol{x}[i] \,:\, 1 \le i \le |\boldsymbol{x}| \,\}$ is the set of all its coordinates. A string is identified with a vector over $\{0, 1\}$, so that if x is a string then $x[i]$ is its i-th bit and $|x|$ is its length. We denote $x[i..j]$ the i-th bit to the j-th bit of string x. By ε we denote the empty vector or string. The size of a set S is denoted $|S|$. For sets D, R let $\mathsf{AF}(D, R)$ denote the set of all functions $f : D \to R$. If $f : D \to R$ is a function then $\mathsf{Img}(f) = \{\, f(x) \,:\, x \in D \,\} \subseteq R$ is its image. We say that f is *regular* if every $y \in \mathsf{Img}(f)$ has the same number of pre-images under f. By $\{0, 1\}^{\le L}$ we denote the set of all strings of length at most L. For any variables a and b, the expression $[[a = b]]$ denotes the Boolean value true when a and b contain the same value and false otherwise.

Let S be a finite set. We let $x \leftarrow\$\, S$ denote sampling an element uniformly at random from S and assigning it to x. We let $y \leftarrow A[\mathrm{O}_1, \ldots](x_1, \ldots; r)$ denote executing algorithm A on inputs $x_1, \ldots$ and coins r with access to oracles $\mathrm{O}_1, \ldots$ and letting y be the result. We let $y \leftarrow\$\, A[\mathrm{O}_1, \ldots](x_1, \ldots)$ be the resulting of picking r at random and letting $y \leftarrow A[\mathrm{O}_1, \ldots](x_1, \ldots; r)$ be the equivalent. We let $\mathrm{OUT}(A[\mathrm{O}_1, \ldots](x_1, \ldots)])$ denote the set of all possible outputs of A when invoked with inputs $x_1, \ldots$ and oracles $\mathrm{O}_1, \ldots$. Algorithms are randomized unless otherwise indicated. Running time is worst case.

GAMES. We use the code-based game playing framework of [10]. (See Fig. 1 for an example.) Games have procedures, also called oracles. Among the oracles are INIT and a FIN. In executing an adversary $\mathcal{A}$ with a game G, the adversary may query the oracles at will. We require that the adversary's first oracle query be to INIT and its last to FIN and it query these oracles at most once. The value return by the FIN procedure is taken as the game output. By $\mathrm{G}(\mathcal{A}) \Rightarrow y$ we denote the event that the execution of game G with adversary $\mathcal{A}$ results in output y. We write $\Pr[\mathrm{G}(\mathcal{A})]$ as shorthand for $\Pr[\mathrm{G}(\mathcal{A}) \Rightarrow \mathsf{true}]$, the probability that the game returns true.

In writing game or adversary pseudocode, it is assumed that Boolean variables are initialized to false, integer variables are initialized to 0 and set-valued variables are initialized to the empty set $\emptyset$.

We adopt the convention that the running time of an adversary is the time for the execution of the game with the adversary, so that the time for oracles to respond to queries is included. In counting the number of queries to an oracle O, we have two metrics. We let $\mathrm{Q}_{\mathrm{O}}^{\mathcal{A}}$ denote the number of queries made to O in the execution of the game with $\mathcal{A}$. (This includes not just queries made directly by $\mathcal{A}$ but also those made by game oracles, the latter usually arising from game executions of scheme algorithms that use O.) In particular, under this metric, the number of queries to a random oracle FO includes those made by scheme algorithms executed by game procedures. With $\mathrm{q}_{\mathrm{O}}^{\mathcal{A}}$ we count only queries made directly by $\mathcal{A}$ to O, not by other game oracles or scheme algorithms. These counts are all worst case.

Groups. Throughout the paper, we fix integers k and b, an odd prime p, and a positive integer f such that $2^{\mathsf{f}} < \mathsf{p}$. We then fix two groups: $\mathbb{G}$, a group of order $\mathsf{p} \cdot 2^{\mathsf{f}}$ whose elements are k-bit strings, and its cyclic subgroup $\mathbb{G}_{\mathsf{p}}$ of order p. We prove in Appendix A that this subgroup is unique, and that it has an efficient membership test. We also assume an efficient membership test for $\mathbb{G}$. We will use additive notation for the group operation, and we let $0_{\mathbb{G}}$ denote the identity element of $\mathbb{G}$. We let $\mathbb{G}_{\mathsf{p}}^* = \mathbb{G} \setminus \{0_{\mathbb{G}}\}$ denote the set of non-identity elements of $\mathbb{G}_{\mathsf{p}}$, which is its set of generators. We fix a distinguished generator $\mathsf{B} \in \mathbb{G}_{\mathsf{p}}^*$. Then for any $X \in \mathbb{G}^*$, the discrete logarithm base B of X is denoted $\mathsf{DL}_{\mathbb{G},\mathsf{B}}(X)$, and it is in the set $\mathbb{Z}_{|\mathbb{G}|}$. The instantiation of $\mathbb{G}$ used in Ed25519 is described in Appendix B.

3 Functor framework

Our treatment relies on the notion of functors [6], which are functions that access an idealized primitive. We give relevant definitions, starting with signature schemes whose security is measured relative to a functor. Then we extend the notions of PRGs and PRFs to functors.

Function spaces. In using the random oracle model [9], works in the literature sometimes omit to say what exactly are the domain and range of the underlying functions, and, when multiple functions are present, whether or not they are independent. (Yet, implicitly their proofs rely on certain choices.) For greater precision, we use the language of function spaces of [6], which we now recall.

A *function space* O is a set of tuples $\mathsf{H} = (\mathsf{H}_1, \ldots, \mathsf{H}_n)$ of functions. The integer n is called the arity of the function space, and can be recovered as $\mathsf{O.arity}$. We view H as taking an input X that it parses as (i, x) to return $\mathsf{H}_i(x)$.

Functors. Following [6], we use the term functor for a transform that constructs one function from another. A functor $\mathbf{F} : \mathsf{SS} \to \mathsf{ES}$ takes as oracle a function h from a starting function space SS and returns a function $\mathbf{F}[\mathsf{h}]$ in the ending function space ES. (The term is inspired by category theory, where a functor maps from one category into another. In our case, the categories are function spaces.) If ES has arity n, then we also refer to n as the arity of $\mathbf{F}$, and write $\mathbf{F}_i$ for the functor which returns the i-th component of $\mathbf{F}$. That is, $\mathbf{F}_i[\mathsf{h}]$ lets $\mathsf{H} \leftarrow \mathbf{F}[\mathsf{h}]$ and returns H_i.

MD functor. We are interested in the Merkle-Damgård [30, 19] transform. This transform constructs a hash function with domain $\{0,1\}^*$ from a compression function $\mathsf{h} : \{0,1\}^{b+2k} \to \{0,1\}^{2k}$ for some integers b and k. The compression function takes a $2k$-bit chaining variable y and a b-bit block B to return a $2k$ bit output $\mathsf{h}(y \| B)$. In the case of SHA512, the hash function used in EdDSA, the compression function sha512 has $b = 1024$ and $k = 256$ (so the chaining variable is 512 bits and a block is 1024 bits), while $b = 512$ and $k = 128$ for SHA256. In our language, the Merkle-Damgård transform is a functor $\mathbf{MD} : \mathsf{AF}(\{0,1\}^{b+2k}, \{0,1\}^{2k}) \to \mathsf{AF}(\{0,1\}^*, \{0,1\}^{2k})$. It is parameterized by a padding function pad that takes the length ℓ of an input to the hash function and returns a padding string such that $\ell + |\mathsf{pad}(\ell)|$ is a multiple of b. Specifically, $\mathsf{pad}(\ell)$ returns $10^*\langle\ell\rangle$ where $\langle\ell\rangle$ is a 64-bit, resp. 128-bit encoding of ℓ for `SHA256` resp. `SHA512`, and 0^* indicates the minimum number p of 0s needed to make $\ell+1+p+64$, resp. $\ell+1+p+128$ a multiple of b. We also fix an "initialization vector" $IV \in \{0,1\}^{2k}$. Given oracle h, the functor defines hash function $\mathsf{H} = \mathbf{MD}[\mathsf{h}] : \{0,1\}^* \to \{0,1\}^{2k}$ as follows:

```
Functor $\mathbf{MD}[\mathsf{h}](X)$
$y[0] \leftarrow IV$
$P \leftarrow \mathsf{pad}(|X|)$ ; $X'[1]\ldots X'[m] \leftarrow X\|P$   // Split $X\|P$ into $b$-bit blocks
For $i = 1,\ldots,m$ do $y[i] \leftarrow \mathsf{h}(y[i-1]\|X'[i])$
Return $y[m]$
```

Strictly speaking, the domain is only strings of length less than 2^{64} resp. 2^{128}, but since this is huge in practice, we view the domain as $\{0,1\}^*$.

Signature scheme syntax. We give an enhanced, flexible syntax for a signature scheme DS. We want to cover ROM schemes, which means scheme algorithms have oracle access to a function H, but of what range and domain? Since these can vary from scheme to scheme, we have the scheme begin by naming the function space $\mathsf{DS.FS}$ from which H is drawn. We see the key-generation algorithm $\mathsf{DS.Kg}$ as first picking a signing key $sk \leftarrow\!\!\$\ \mathsf{DS.SK}$ via a signing-key generation algorithm $\mathsf{DS.SK}$, then obtaining the public verification key $vk \leftarrow \mathsf{DS.PK}[\mathsf{H}](sk)$ by applying a deterministic verification-key generation algorithm $\mathsf{DS.PK}$, and finally returning (vk, sk). (For simplicity, $\mathsf{DS.SK}$, unlike other scheme algorithms, does not have access to H.) We break it up like this because we may need to explicitly refer to the sub-algorithms in constructions. Continuing, via $\sigma \leftarrow \mathsf{DS.Sign}[\mathsf{H}](sk, vk, M; r)$ the signing algorithm takes sk, vk, a message $M \in \{0,1\}^*$, and randomness r from the randomness space $\mathsf{DS.SR}$ of the algorithm, to return a signature σ. As usual, $\sigma \leftarrow\!\!\$\ \mathsf{DS.Sign}[\mathsf{H}](sk, vk, M)$ is shorthand for picking $r \leftarrow\!\!\$\ \mathsf{DS.SR}$ and returning $\sigma \leftarrow \mathsf{DS.Sign}[\mathsf{H}](sk, vk, M; r)$. Via $b \leftarrow \mathsf{DS.Vf}[\mathsf{H}](vk, M, \sigma)$, the verification algorithm obtains a boolean decision $b \in \{\mathsf{true}, \mathsf{false}\}$ about the validity of the signature. The correctness requirement is that for all $\mathsf{H} \in \mathsf{DS.FS}$, all $(vk, sk) \in \mathrm{OUT}(\mathsf{DS.Kg}[\mathsf{H}])$, all $M \in \{0,1\}^*$ and all $\sigma \in \mathrm{OUT}(\mathsf{DS.Sign}[\mathsf{H}](sk, vk, M))$ we have $\mathsf{DS.Vf}[\mathsf{H}](vk, M, \sigma) = \mathsf{true}$.

UF security. We want to discuss security of a signature scheme DS under different ways in which the functions in $\mathsf{DS.FS}$ are chosen or built. Game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{FF}}$ in Fig. 1 is thus parameterized by a functor $\mathbf{FF} : \mathsf{SS} \to \mathsf{DS.FS}$. At line 1, a starting function h is chosen from the starting space of the functor, and then the function $\mathsf{H} \in \mathsf{DS.FS}$ that the scheme algorithms (key-generation, signing and verification) get as oracle is determined as $\mathsf{H} \leftarrow \mathbf{FF}[\mathsf{h}]$. The adversary, however, via oracle FO, gets access to h, which here is the random oracle. The rest is as per the usual unforgeability definition. (Given in the standard model in [23] and extended to the ROM in [9].) We define the UF advantage of adversary $\mathcal{A}$ as $\mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{FF}}(\mathcal{A}) = \Pr[\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{FF}}(\mathcal{A})]$.

PRGs and PRFs. The usual definition of a PRGs is for a function; we define it instead for a functor $\mathbf{P}$. The game $\mathbf{G}^{\mathrm{prg}}_{\mathbf{P}}$ is in Figure 1. It picks a function h from the starting space SS of the functor. The functor now determines a function $\mathbf{P}[\mathsf{h}] : \{0,1\}^k \to \{0,1\}^\ell$. The game then follows the usual PRG one for this function, additionally giving the adversary oracle access to h via oracle FO. We let $\mathbf{Adv}^{\mathrm{prg}}_{\mathbf{P}}(\mathcal{A}) = 2\Pr[\mathbf{G}^{\mathrm{prg}}_{\mathbf{P}}(\mathcal{A})] - 1$.

Similarly we extend the usual definition of PRG security to a functor $\mathbf{F}$, via game $\mathbf{G}^{\mathrm{prf}}_{\mathbf{F}}$ of Figure 1. Here, for h in the starting space SS of the functor, the defined function maps as $\mathbf{F}[\mathsf{h}] : \{0,1\}^k \times \{0,1\}^* \to R$ for some k and range set R. We let $\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{A}) = 2\Pr[\mathbf{G}^{\mathrm{prf}}_{\mathbf{F}}(\mathcal{A})] - 1$.

```
Game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{FF}}$

INIT:
  $\mathsf{h} \leftarrow\!\$\ \mathsf{SS}$ ; $\mathsf{H} \leftarrow \mathbf{FF}[\mathrm{FO}]$ ; $(vk, sk) \leftarrow\!\$\ \mathsf{DS.Kg}[\mathsf{H}]$ ; Return $vk$

SIGN($M$):
  $\sigma \leftarrow\!\$\ \mathsf{DS.Sign}[\mathsf{H}](sk, vk, M)$ ; $S \leftarrow S \cup \{M\}$ ; Return $\sigma$

FO($X$):
  Return $\mathsf{h}(X)$

FIN($M_*, \sigma_*$):
  If $(M_* \in S)$ then return false
  Return $\mathsf{DS.Vf}[\mathsf{H}](vk, M_*, \sigma_*)$
```

```
Game $\mathbf{G}^{\mathrm{prg}}_{\mathbf{P}}$

INIT:
  $\mathsf{h} \leftarrow\!\$\ \mathsf{SS}$ ; $c \leftarrow\!\$\ \{0,1\}$
  $s \leftarrow\!\$\ \{0,1\}^k$ ; $y_1 \leftarrow \mathbf{P}[\mathrm{FO}](s)$
  $y_0 \leftarrow\!\$\ \{0,1\}^\ell$
  Return $y_c$

FO($X$):
  Return $\mathsf{h}(X)$

FIN($c'$):
  Return $(c = c')$
```

```
Game $\mathbf{G}^{\mathrm{prf}}_{\mathbf{F}}$

INIT:
  $\mathsf{h} \leftarrow\!\$\ \mathsf{SS}$ ; $c \leftarrow\!\$\ \{0,1\}$ ; $K \leftarrow\!\$\ \{0,1\}^k$

FN($X$):
  If $\mathrm{YT}[X] \neq \bot$ then
    If $(c = 1)$ then $\mathrm{YT}[X] \leftarrow \mathbf{F}[\mathrm{FO}](K, X)$
    Else $\mathrm{YT}[X] \leftarrow\!\$\ R$
  Return $\mathrm{YT}[X]$

FO($X$):
  Return $\mathsf{h}(X)$

FIN($c'$):
  Return $(c = c')$
```

Figure 1: Top: Game defining UF security of signature scheme DS relative to functor $\mathbf{FF} : \mathsf{SS} \to \mathsf{DS.FS}$. Bottom Left: Game defining PRG security of functor $\mathbf{P} : \mathsf{SS} \to \mathsf{AF}(\{0,1\}^k, \{0,1\}^\ell)$. Bottom Right: Game defining PRF security of functor $\mathbf{F} : \mathsf{SS} \to \mathsf{AF}(\{0,1\}^k \times \{0,1\}^*, R)$.

4 The soundness of Derive-then-Derandomize

We specify a general signature-hardening transform that we call Derive-then-Derandomize (**DtD**) and prove that it preserves the security of the starting signature scheme.

THE **DtD** TRANSFORM. Let DS be a given signature scheme that we call the base signature scheme. It will be the (general) Schnorr scheme in our application. Assume for simplicity that its function space $\mathsf{DS.FS}$ has arity 1.

The **DtD** (derive then de-randomize) transform constructs a signature scheme $\overline{\mathsf{DS}} = \mathbf{DtD}[\mathsf{DS}, \mathtt{CF}]$ based on DS and a function $\mathtt{CF} : \{0,1\}^k \to \mathrm{OUT}(\mathsf{DS.SK})$, called the clamping function, that turns a k-bit string into a signing key for DS. The algorithms of $\overline{\mathsf{DS}}$ are shown in Figure 2. They have access to oracle H that specifies sub-functions $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$. Function $\mathsf{H}_1 : \{0,1\}^k \to \{0,1\}^{2k}$ expands the signing key $\overline{sk}$ of $\overline{\mathsf{DS}}$ into sub-keys e_1 and e_2. The clamping function is applied to e_1 to get a signing key for the base scheme, and its associated verification key is returned as the one for the

```
$\overline{\mathsf{DS}}.\mathsf{SK}$:
1 $\overline{sk} \leftarrow\!\!\$\ \{0,1\}^k$ ; Return $\overline{sk}$
$\overline{\mathsf{DS}}.\mathsf{PK}[\mathsf{H}](\overline{sk})$:
2 $e_1 \| e_2 \leftarrow \mathsf{H}_1(\overline{sk})$ ; $sk \leftarrow \mathtt{CF}(e_1)$
3 $vk \leftarrow \mathsf{DS.PK}[\mathsf{H}_3](sk)$
4 Return $vk$
$\overline{\mathsf{DS}}.\mathsf{Sign}[\mathsf{H}](\overline{sk}, vk, M)$:
5 $e_1 \| e_2 \leftarrow \mathsf{H}_1(\overline{sk})$ ; $sk \leftarrow \mathtt{CF}(e_1)$
6 $r \leftarrow \mathsf{H}_2(e_2, M)$
7 $\sigma \leftarrow \mathsf{DS.Sign}[\mathsf{H}_3](sk, vk, M; r)$
8 Return $\sigma$
$\overline{\mathsf{DS}}.\mathsf{Vf}[\mathsf{H}](vk, M, \sigma)$:
9 Return $\mathsf{DS.Vf}[\mathsf{H}_3](vk, M, \sigma)$
```

```
$\mathsf{DS}^*.\mathsf{SK}$:
1 $\overline{sk} \leftarrow\!\!\$\ \{0,1\}^k$ ; Return $\overline{sk}$
$\mathsf{DS}^*.\mathsf{PK}[\mathsf{G}](\overline{sk})$:
2 $sk \leftarrow \mathtt{CF}(\overline{sk})$
3 $vk \leftarrow \mathsf{DS.PK}[\mathsf{G}](sk)$
4 Return $vk$
$\mathsf{DS}^*.\mathsf{Sign}[\mathsf{G}](\overline{sk}, vk, M)$:
5 $sk \leftarrow \mathtt{CF}(\overline{sk})$
6 $\sigma \leftarrow\!\!\$\ \mathsf{DS.Sign}[\mathsf{G}](sk, vk, M)$
7 Return $\sigma$
$\mathsf{DS}^*.\mathsf{Vf}[\mathsf{G}](vk, M, \sigma)$:
8 Return $\mathsf{DS.Vf}[\mathsf{G}](vk, M, \sigma)$
```

Figure 2: **Left:** The signature scheme $\overline{\mathsf{DS}} = \mathbf{DtD}[\mathsf{DS}, \mathtt{CF}]$ constructed by the **DtD** transform applied to signature scheme DS and clamping function $\mathtt{CF}: \{0,1\}^k \to \mathrm{OUT}(\mathsf{DS.SK})$. **Right:** The signature scheme $\overline{\mathsf{DS}} = \mathbf{JCl}[\mathsf{DS}, \mathtt{CF}]$ constructed by the **JCl** transform.

new scheme at line 4. At line 6, function $\mathsf{H}_2 : \{0,1\}^k \times \{0,1\}^* \to \mathsf{DS.SR}$ is applied to the second sub-key e_2 and the message M to determine signing randomness r for the line 5 invocation of the base signing algorithm. Finally, $\mathsf{H}_3 \in \mathsf{DS.FS}$ is an oracle for the algorithms of DS. Formally the oracle space $\overline{\mathsf{DS}}.\mathsf{FS}$ of $\overline{\mathsf{DS}}$ is the arity 3 space consisting of all $\mathsf{H} = (\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3)$ that map as above.

Viewing the PRG H_1, PRF H_2 and oracle H_3 for the base scheme as specified in the function space is convenient for our application to EdDSA, where they are all based on **MD** with the *same* underlying idealized compression function.

Just clamp. Given a signature scheme DS and a clamping function $\mathtt{CF}: \{0,1\}^k \to \mathrm{OUT}(\mathsf{DS.SK})$, it is useful to also consider the signature scheme $\mathsf{DS}^* = \mathbf{JCl}[\mathsf{DS}, \mathtt{CF}]$ that does just the clamping. The scheme is shown in Figure 2. Its oracle space is the same as that of DS and is assumed to have arity 1. On the right of Figure 2 the function drawn from it is denoted G; it will be the same as H_3 on the left.

Security of **DtD**. We study the security of the scheme $\overline{\mathsf{DS}} = \mathbf{DtD}[\mathsf{DS}, \mathtt{CF}]$ obtained via the **DtD** transform.

When we prove security of $\overline{\mathsf{DS}}$, it will be with respect to a functor **FF** that constructs all of $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$. This means that these three functions could all depend on the same starting function that **FF** uses, and in particular not be independent of each other. An important element of the following theorem is that it holds even in this case, managing to reduce security to conditions on the individual functors despite their using related (in fact, the same) underlying starting function.

Theorem 4.1 *Let* DS *be a signature scheme. Let* $\mathtt{CF}: \{0,1\}^k \to \mathrm{OUT}(\mathsf{DS.SK})$ *be a clamping function. Let* $\overline{\mathsf{DS}} = \mathbf{DtD}[\mathsf{DS}, \mathtt{CF}]$ *and* $\mathsf{DS}^* = \mathbf{JCl}[\mathsf{DS}, \mathtt{CF}]$ *be the signature schemes obtained by the above transforms. Let* $\mathbf{FF} : \mathrm{SS} \to \overline{\mathsf{DS}}.\mathsf{FS}$ *be a functor that constructs the function* H *that algorithms of* $\overline{\mathsf{DS}}$ *use as an oracle. Let* $\mathcal{A}$ *be an adversary attacking the* $\mathbf{G}^{\mathrm{uf}}$ *security of* $\overline{\mathsf{DS}}$. *Then there are adversaries* $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ *such that*

$$\mathbf{Adv}^{\mathrm{uf}}_{\overline{\mathsf{DS}},\mathbf{FF}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prg}}_{\mathbf{FF}_1}(\mathcal{A}_1) + \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{FF}_2}(\mathcal{A}_2) + \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{FF}_3}(\mathcal{A}_3) .$$

```
Games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$

INIT:
1  $\mathsf{h} \leftarrow\!\!\$\; \mathsf{SS}$
2  $\overline{sk} \leftarrow\!\!\$\; \{0,1\}^k$ ; $\mathrm{e}_1\|\mathrm{e}_2 \leftarrow \mathbf{FF}_1[\mathrm{FO}](\overline{sk})$ // Game $\mathrm{G}_0$
3  $\mathrm{e}_1\|\mathrm{e}_2 \leftarrow\!\!\$\; \{0,1\}^{2k}$ // Games $\mathrm{G}_1, \mathrm{G}_2$
4  $sk \leftarrow \mathtt{CF}(\mathrm{e}_1)$ ; $vk \leftarrow \mathsf{DS.PK}[\mathbf{FF}_3[\mathrm{FO}]](sk)$ ; Return $vk$

SIGN($M$):
5  If $\mathrm{ST}[M] \neq \perp$ then return $\mathrm{ST}[M]$
6  $r \leftarrow \mathbf{FF}_2[\mathrm{FO}](e_2, M)$ // Games $\mathrm{G}_0, \mathrm{G}_1$
7  $r \leftarrow\!\!\$\; \mathsf{DS.SR}$ // Game $\mathrm{G}_2$
8  $\mathrm{ST}[M] \leftarrow \mathsf{DS.Sign}[\mathbf{FF}_3[\mathrm{FO}]](sk, vk, M; r)$ ; Return $\mathrm{ST}[M]$

FO($X$):
9  Return $\mathsf{h}(X)$

FIN($M_*, \sigma_*$):
10 If ($\mathrm{ST}[M_*] \neq \perp$) then return false
11 Return $\mathsf{DS.Vf}[\mathbf{FF}_3[\mathrm{FO}]](vk, M_*, \sigma_*)$
```

Figure 3: Games for proof of Theorem 4.1. A line annotated with names of games is included only in those games.

The constructed adversaries have $\mathrm{Q}^{\mathcal{A}_i}_{\mathrm{FO}} = \mathrm{Q}^{\mathcal{A}}_{\mathrm{FO}}$ *(*$i = 1, 2, 3$*) and approximately the same running time as* $\mathcal{A}$*. Adversary* $\mathcal{A}_2$ *makes* $\mathrm{Q}^{\mathcal{A}}_{\mathrm{SIGN}}$ *queries to* FN*. Adversary* $\mathcal{A}_3$ *makes* $\mathrm{Q}^{\mathcal{A}}_{\mathrm{SIGN}}$ *queries to* SIGN*.*

Recall that $\mathrm{Q}^{\mathcal{B}}_{\mathrm{O}}$ means the number of queries made to oracle O in the execution of the game with adversary $\mathcal{B}$, so queries made by scheme algorithms, run in the game in response to $\mathcal{B}$'s queries, are included. The theorem says the number of queries to FO is preserved under this metric. The number of direct queries to FO is not necessarily preserved. Thus $\mathrm{q}^{\mathcal{A}_i}_{\mathrm{FO}}$ could be more than $\mathrm{q}^{\mathcal{A}}_{\mathrm{FO}}$. For example $\mathrm{q}^{\mathcal{A}_1}_{\mathrm{FO}}$ is $\mathrm{q}^{\mathcal{A}}_{\mathrm{FO}}$ plus the number of queries to FO made by the calls to $\mathbf{FF}_3[\mathrm{FO}]$, the latter calls in turn made by the execution of $\mathsf{DS.Sign}[\mathbf{FF}_3[\mathrm{FO}]]$ across the different queries to SIGN. Accounting precisely for this is involved, whence a preference where possible for the game-inclusive query metric $\mathrm{Q}^{\cdot}_{\cdot}$.

Proof of Theorem 4.1: The proof uses code-based game playing [10]. Consider the games of Figure 3. Let $\epsilon_i = \Pr[\mathrm{G}_i(\mathcal{A})]$ for $i = 0, 1, 2$.

Game G_0 is the $\mathbf{G}^{\mathrm{uf}}$ game for $\overline{\mathsf{DS}}$ except that the signature of M is stored in table ST at line 8, and, at line 5, if a signature for M already exists, it is returned directly. Since signing in $\overline{\mathsf{DS}}$ is deterministic, meaning the signature is always the same for a given message and signing key, this does not change what SIGN returns, and thus

$$\begin{aligned}\mathbf{Adv}^{\mathrm{uf}}_{\overline{\mathsf{DS}},\mathbf{FF}}(\mathcal{A}) &= \epsilon_0 \\ &= (\epsilon_0 - \epsilon_1) + (\epsilon_1 - \epsilon_2) + \epsilon_2 \ .\end{aligned}$$

We bound each of the three terms above in turn.

The change in moving to game G_1 is at line 3, where we sample $e_1\|e_2$ uniformly from the set $\{0,1\}^{2k}$ rather than obtaining it via $\mathbf{FF}_1[\mathrm{FO}]$ as in game G_0. We build PRG adversary $\mathcal{A}_1$ such that

$$\epsilon_0 - \epsilon_1 \le \mathbf{Adv}^{\mathrm{prg}}_{\mathbf{FF}_1}(\mathcal{A}_1) \ . \qquad (1)$$

Adversary $\mathcal{A}_1$ is playing game $\mathbf{G}^{\mathrm{prg}}_{\mathbf{FF}_1}$. It gets its challenge via $e_1 \| e_2 \leftarrow \mathbf{G}^{\mathrm{prg}}_{\mathbf{FF}_1}.\textsc{Init}$. It lets $sk \leftarrow \mathtt{CF}(e_1)$ and $vk \leftarrow \mathsf{DS.PK}[\mathbf{FF}_3[\mathbf{G}^{\mathrm{prg}}_{\mathbf{FF}_1}.\mathrm{FO}]](sk)$ where $\mathbf{G}^{\mathrm{prg}}_{\mathbf{FF}_1}.\mathrm{FO}$ is the oracle provided in its own game. It runs $\mathcal{A}$, returning vk in response to $\mathcal{A}$'s $\textsc{Init}$ query. It answers $\textsc{Sign}$ queries as do $\mathrm{G}_0, \mathrm{G}_1$ except that it uses $\mathbf{G}^{\mathrm{prg}}_{\mathbf{FF}_1}.\mathrm{FO}$ in place of FO at lines 6,8. As part of this simulation, it maintains table ST. It answers FO queries via $\mathbf{G}^{\mathrm{prg}}_{\mathbf{FF}_1}.\mathrm{FO}$. When $\mathcal{A}$ calls $\textsc{Fin}(M_*, \sigma_*)$, adversary $\mathcal{A}_1$ lets $c' \leftarrow 1$ if $\mathsf{DS.Vf}[\mathbf{FF}_3[\mathbf{G}^{\mathrm{prg}}_{\mathbf{FF}_1}.\mathrm{FO}]](vk, M_*, \sigma_*)$ is true and $\mathrm{ST}[M_*] = \perp$, and otherwise lets $c' \leftarrow 0$. It then calls $\mathbf{G}^{\mathrm{prg}}_{\mathbf{FF}_1}.\textsc{Fin}(c')$. When the challenge bit c in game $\mathbf{G}^{\mathrm{prg}}_{\mathbf{FF}_1}$ is $c = 1$, the view of $\mathcal{A}$ is as in G_0, and when $c = 0$ it is as in G_1, which explains Eq. (1).

Moving to G_2, the change is that line 6 is replaced by line 7, meaning signing coins are now chosen at random from the randomness space $\mathsf{DS.SR}$ of DS. We build PRF adversary $\mathcal{A}_2$ such that

$$\epsilon_1 - \epsilon_2 \le \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{FF}_2}(\mathcal{A}_2) \ . \qquad (2)$$

Adversary $\mathcal{A}_2$ is playing game $\mathbf{G}^{\mathrm{prf}}_{\mathbf{FF}_2}$. It picks $e_1 \| e_2 \leftarrow\$ \{0,1\}^{2k}$. It lets $sk \leftarrow \mathtt{CF}(e_1)$ and $vk \leftarrow \mathsf{DS.PK}[\mathbf{FF}_3[\mathbf{G}^{\mathrm{prf}}_{\mathbf{FF}_2}.\mathrm{FO}]](sk)$ where $\mathbf{G}^{\mathrm{prg}}_{\mathbf{FF}_2}.\mathrm{FO}$ is the oracle provided in its own game. It runs $\mathcal{A}$, returning vk in response to $\mathcal{A}$'s $\textsc{Init}$ query. It answers $\textsc{Sign}$ queries as does G_1 except that it uses $\mathbf{G}^{\mathrm{prf}}_{\mathbf{FF}_2}.\mathrm{FN}$ in place of $\mathbf{FF}_2[\mathrm{FO}]$ at line 6 and $\mathbf{G}^{\mathrm{prf}}_{\mathbf{FF}_2}.\mathrm{FO}$ in place of FO in line 8. As part of this simulation, it maintains table ST. It answers FO queries via $\mathbf{G}^{\mathrm{prf}}_{\mathbf{FF}_2}.\mathrm{FO}$. When $\mathcal{A}$ calls $\textsc{Fin}(M_*, \sigma_*)$, adversary $\mathcal{A}_2$ lets $c' \leftarrow 1$ if $\mathsf{DS.Vf}[\mathbf{FF}_3[\mathbf{G}^{\mathrm{prf}}_{\mathbf{FF}_2}.\mathrm{FO}]](vk, M_*, \sigma_*)$ is true and $\mathrm{ST}[M_*] = \perp$, and otherwise lets $c' \leftarrow 0$. It then calls $\mathbf{G}^{\mathrm{prf}}_{\mathbf{FF}_2}.\textsc{Fin}(c')$. When the challenge bit c in game $\mathbf{G}^{\mathrm{prf}}_{\mathbf{FF}_2}$ is $c = 1$, the view of $\mathcal{A}$ is as in G_1, and when $c = 0$ it is as in G_2, which explains Eq. (2).

Finally we build adversary $\mathcal{A}_3$ such that

$$\epsilon_2 \le \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*, \mathbf{FF}_3}(\mathcal{A}_3) \ . \qquad (3)$$

Adversary $\mathcal{A}_3$ is playing game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}^*, \mathbf{FF}_3}$. It lets $vk \leftarrow \mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}^*, \mathbf{FF}_3}.\textsc{Init}$. It runs $\mathcal{A}$, returning vk in response to $\mathcal{A}$'s $\textsc{Init}$ query. When $\mathcal{A}$ makes query M to $\textsc{Sign}$, it answers as per the following:

```
If $\mathrm{ST}[M] \neq \perp$ then return $\mathrm{ST}[M]$
$\mathrm{ST}[M] \leftarrow\$ \mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}^*, \mathbf{FF}_3}.\textsc{Sign}(M)$ ; Return $\mathrm{ST}[M]$
```

Note that memoizing signatures in ST is important here to ensure that the $\textsc{Sign}$ queries of $\mathcal{A}$ are correctly simulated. It answers FO queries via $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}^*, \mathbf{FF}_3}.\mathrm{FO}$. When $\mathcal{A}$ calls $\textsc{Fin}(M_*, \sigma_*)$, adversary $\mathcal{A}_2$ calls $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}^*, \mathbf{FF}_3}.\textsc{Fin}(M_*, \sigma_*)$. The distribution of signatures that $\mathcal{A}$ is given, and of the keys underlying them, is as in G_2, which explains Eq. (3).

Note that the constructed adversaries having access to oracle FO in their games is important to their ability to simulate $\mathcal{A}$ faithfully.

With regard to the costs (number of queries, running time) of the constructed adversaries, recall that we have defined these as the costs in the execution of the adversary with the game that the adversary is playing, so for example the number of queries to FO includes the ones made by algorithms executed in the game. When this is taken into account, queries to FO are preserved, and the other claims are direct. ▌

Security of **JCl**. We have now reduced the security of $\overline{\mathsf{DS}}$ to that of DS^*. To further reduce the security of DS^* to that of DS, we give a general result on clamping. Let $\mathcal{K} = \mathrm{OUT}(\mathsf{DS.SK})$ and let $\mathtt{CF}: \{0,1\}^k \to \mathcal{K}$ be a clamping function. As per terminology in Section 2, recall that

$\mathsf{Img}(\mathtt{CF}) = \{ \mathtt{CF}(\overline{sk}) : |\overline{sk}| = k \} \subseteq \mathcal{K}$ is the image of the clamping function, and CF is regular if every $y \in \mathsf{Img}(\mathtt{CF})$ has the same number of pre-images under CF.

Theorem 4.2 *Let* DS *be a signature scheme such that* DS.SK *draws its signing key* $sk \leftarrow\$ \mathcal{K}$ *at random from a set* $\mathcal{K}$*. Let* $\mathtt{CF} : \{0,1\}^k \to \mathcal{K}$ *be a regular clamping function. Let* $\delta = |\mathsf{Img}(\mathtt{CF})|/|\mathcal{K}| > 0$*. Let* $\mathsf{DS}^* = \mathbf{JCl}[\mathsf{DS}, \mathtt{CF}]$ *be the signature scheme obtained by the just-clamp transform. Let* $\mathbf{FF} : \mathsf{SS} \to \mathsf{DS.FS}$ *be any functor. Let* $\mathcal{B}$ *be an adversary attacking the* $\mathbf{G}^{\mathrm{uf}}$ *security of* DS^**. Then*

$$\mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{FF}}(\mathcal{B}) \leq (1/\delta) \cdot \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{FF}}(\mathcal{B}) \; .$$

Proof of Theorem 4.2: We consider running $\mathcal{B}$ in game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{FF}}$, where the signing key is $sk \leftarrow\$ \mathcal{K}$. With probability δ we have $sk \in \mathsf{Img}(\mathtt{CF})$. Due to the regularity of CF, key sk now has the same distribution as a key $\mathtt{CF}(\overline{sk})$ for $\overline{sk} \leftarrow\$ \{0,1\}^k$ drawn in game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{FF}}$. Thus $\mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{FF}}(\mathcal{B}) \geq \delta \cdot \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{FF}}(\mathcal{B})$. ▌

5 Security of EdDSA

The Schnorr scheme. Let the prime-order group $\mathbb{G}_{\mathsf{p}}$ of k-bit strings with generator $\mathtt{B}$ be as described in Section 2. The algorithms of the Schnorr signature scheme $\mathsf{DS} = \mathsf{Sch}$ are shown on the left in Figure 4. The function space DS.FS is $\mathsf{AF}(\{0,1\}^*, \mathbb{Z}_{\mathsf{p}})$. (Implementations may use a hash function that outputs a string and embed the result in $\mathbb{Z}_{\mathsf{p}}$ but following prior proofs [1] we view the hash function as directly mapping into $\mathbb{Z}_{\mathsf{p}}$.) Verification is parameterized by an algorithm VF to allow us to consider strict and permissive verification in a modular way. The corresponding choices of verification algorithms are at the bottom of Figure 4. The signing randomness space is $\mathsf{DS.SR} = \mathbb{Z}_{\mathsf{p}}$.

Schnorr signatures have a few variants that differ in details. In Schnorr's paper [39], the challenge is $c = \mathsf{H}(\mathtt{R} \| M) \bmod \mathsf{p}$. Our inclusion of the public key in the input to H follows Bernstein [12] and helps here because it is what EdDSA does. It doesn't affect security. (The security of the scheme that includes the public key in the hash input is implied by the security of the one that doesn't via a reduction that includes the public key in the message.) Also in [39], the signature is (c, z). The version we use, where it is $(\mathtt{R}, z)$, is from [1]. However, BBSS [2] shows that these versions have equivalent security.

The EdDSA scheme. Let the prime-order group $\mathbb{G}_{\mathsf{p}}$ of k-bit strings with generator $\mathtt{B}$ be as before and assume $2^{k-5} < \mathsf{p} < 2^k$. Let $\mathtt{CF} : \{0,1\}^k \to \mathbb{Z}_{\mathsf{p}}$ be the clamping function shown at the bottom of Figure 4. The algorithms of the scheme $\overline{\mathsf{DS}}$ are shown on the right side of Figure 4. The key length is k. As before, the verification algorithm VF is a parameter. The H available to the algorithms defines three sub-functions. The first, $\mathsf{H}_1 : \{0,1\}^k \to \{0,1\}^{2k}$, is used at lines 2,4, where its output is parsed into k-bit halves. The second, $\mathsf{H}_2 : \{0,1\}^k \times \{0,1\}^* \to \mathbb{Z}_{\mathsf{p}}$, is used at line 5 for de-randomization. The third, $\mathsf{H}_3 : \{0,1\}^* \to \mathbb{Z}_{\mathsf{p}}$, plays the role of the function H for the Schnorr schemes. Formally, $\overline{\mathsf{DS}}.\mathsf{FS}$ is the arity-3 function space consisting of all H mapping as just indicated.

In [13, 15], the output of the clamping is an integer that (in our notation) is in the range $2^{k-2}, \ldots, 2^{k-1} - 8$. When used in the scheme, however, it is (implicitly) modulo p. It is convenient for our analysis, accordingly, to define CF to be the result modulo p of the actual clamping. Note that in EdDSA the prime p has magnitude a little more than 2^{k-4} and less than 2^{k-3}.

There are several versions of EdDSA depending on the choice for verification algorithms: strict, permissive or batch VF. We specify the first two choices in Figure 4. Our results hold for all choices

```
DS.SK:
  $s \leftarrow\$ \mathbb{Z}_{\mathsf{p}}$
  Return $s$
DS.PK($s$):
  $\mathtt{A} \leftarrow s \cdot \mathtt{B}$ ; Return $\mathtt{A}$
DS.Sign[H]($s, \mathtt{A}, M$):
  $r \leftarrow\$ \mathbb{Z}_{\mathsf{p}}$ ; $\mathtt{R} \leftarrow r \cdot \mathtt{B}$
  $c \leftarrow \mathsf{H}(\mathtt{R} \| \mathtt{A} \| M)$
  $z \leftarrow (sc + r) \mod \mathsf{p}$
  Return $(\mathtt{R}, z)$
DS.Vf[H]($\mathtt{A}, M, \sigma$):
  $(\mathtt{R}, z) \leftarrow \sigma$
  $c \leftarrow \mathsf{H}(\mathtt{R} \| \mathtt{A} \| M)$
  Return VF($\mathtt{A}, \mathtt{R}, c, z$)
```

```
$\overline{\mathsf{DS}}$.SK:
  $sk \leftarrow\$ \{0,1\}^k$ ; Return $sk$
$\overline{\mathsf{DS}}$.PK($sk$):
  $e_1 \| e_2 \leftarrow \mathsf{H}_1(sk)$ ; $s \leftarrow \mathrm{CF}(e_1)$
  $\mathtt{A} \leftarrow s \cdot \mathtt{B}$ ; Return $\mathtt{A}$
$\overline{\mathsf{DS}}$.Sign[H]($sk, \mathtt{A}, M$):
  $e_1 \| e_2 \leftarrow \mathsf{H}_1(sk)$ ; $s \leftarrow \mathrm{CF}(e_1)$
  $r \leftarrow \mathsf{H}_2(e_2, M)$ ; $\mathtt{R} \leftarrow r \cdot \mathtt{B}$
  $c \leftarrow \mathsf{H}_3(\mathtt{R} \| \mathtt{A} \| M)$
  $z \leftarrow (sc + r) \bmod \mathsf{p}$
  Return $(\mathtt{R}, z)$
$\overline{\mathsf{DS}}$.Vf[H]($\mathtt{A}, M, \sigma$):
  $(\mathtt{R}, z) \leftarrow \sigma$
  $c \leftarrow \mathsf{H}_3(\mathtt{R} \| \mathtt{A} \| M) \bmod \mathsf{p}$
  Return VF($\mathtt{A}, \mathtt{R}, c, z$)
```

```
CF($e$) // $e \in \{0,1\}^k$:
  $t \leftarrow 2^{k-2}$
  for $i \in [4..k-2]$
      $t \leftarrow t + 2^{i-1} \cdot e[i]$
  $s \leftarrow t \bmod \mathsf{p}$
  return $s$
```

```
sVF($\mathtt{A}, \mathtt{R}, c, z$):
  Return $(z \cdot \mathtt{B} = c \cdot \mathtt{A} + \mathtt{R})$

pVF($\mathtt{A}, \mathtt{R}, c, z$):
  Return $2^{\mathsf{f}}(z \cdot \mathtt{B}) = 2^{\mathsf{f}}(c \cdot \mathtt{A} + \mathtt{R})$
```

Figure 4: **Top Left:** the Schnorr scheme. **Top Right:** The EdDSA scheme. **Bottom Left:** EDDSA clamping function (generalized for any k; in the original definition, $k = 256$). **Bottom Right:** Strict and Permissive verification algorithms as choices for VF.

of VF, meaning EdDSA is secure with respect to VF assuming Schnorr is secure with respect to VF. It is in order to make this general claim that we abstract out VF.

Security of EdDSA with independent ROs. As a warm-up, we show security of EdDSA when the three functions it uses are independent random oracles, the setting assumed by BCJZ [15]. However, while they assume hardness of DL, our result is more general, assuming only security of Schnorr with a monolithic random oracle. We can then use known results on Schnorr [36, 1] to recover the result of BCJZ [15], but the proof is simpler and more modular. Also, other known results on Schnorr [38, 5, 21] can be applied to get better bounds. Following this, we will turn to the "real" case, where the three functions are all **MD** with a random compression function.

The Theorem below is for a general prime $\mathsf{p} > 2^{k-5}$ but in EdDSA the prime is $2^{k-4} < \mathsf{p} < 2^{k-3}$ so the value of δ below is $\delta = 2^{k-5}/\mathsf{p} > 2^{k-5}/2^{k-3} = 1/4$, so the factor $1/\delta$ is ≤ 4. We capture the three functions of EdDSA being independent random oracles by setting functor $\mathbf{P}$ below to the identity functor, and similarly capture Schnorr being with a monolithic random oracle by setting $\mathbf{R}$ to be the identity functor.

Theorem 5.1 *Let* $\mathsf{DS} = \mathsf{Sch}$ *be the* Schnorr *signature scheme of Figure 4. Let* $\mathtt{CF} : \{0,1\}^k \to \mathbb{Z}_{\mathsf{p}}$ *be the clamping function of Figure 4. Assume* $\mathsf{p} > 2^{k-5}$ *and let* $\delta = 2^{k-5}/\mathsf{p}$*. Let* $\overline{\mathsf{DS}} = \mathbf{DtD}[\mathsf{DS}, \mathtt{CF}]$

```
Functor $\mathbf{S}_1[\mathsf{h}](sk)$: // $|sk| = k$
  $e \leftarrow \mathbf{MD}[\mathsf{h}](sk)$ ; Return $e$ // $|e| = 2k$
Functor $\mathbf{S}_2[\mathsf{h}](e_2, M)$: // $|e_2| = k$
  Return $\mathbf{MD}[\mathsf{h}](e_2 \| M) \bmod \mathsf{p}$
Functor $\mathbf{S}_3[\mathsf{h}](X)$: // also called $\mathbf{Mod\text{-}MD}$
  Return $\mathbf{MD}[\mathsf{h}](X) \bmod \mathsf{p}$
```

Figure 5: The arity-3 functor $\mathbf{S}$ for EdDSA. Here $\mathsf{h} : \{0,1\}^{b+2k} \to \{0,1\}^{2k}$ is a compression function.

be the EdDSA *signature scheme. Let* $\mathbf{R} : \mathsf{AF}(\{0,1\}^*, \mathbb{Z}_{\mathsf{p}}) \to \mathsf{AF}(\{0,1\}^*, \mathbb{Z}_{\mathsf{p}})$ *be the identity functor. Let* $\mathbf{P} : \overline{\mathsf{DS}}.\mathsf{FS} \to \overline{\mathsf{DS}}.\mathsf{FS}$ *be the identity functor. Let* $\mathcal{A}$ *be an adversary attacking the* $\mathbf{G}^{\mathrm{uf}}$ *security of* $\overline{\mathsf{DS}}$*. Then there is an adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{\mathrm{uf}}_{\overline{\mathsf{DS}},\mathbf{P}}(\mathcal{A}) \le (1/\delta) \cdot \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{R}}(\mathcal{B}) + \frac{2 \cdot \mathrm{Q}^{\mathcal{A}}_{\mathrm{FO}}}{2^k} \ .$$

Adversary $\mathcal{B}$ *preserves the queries and running time of* $\mathcal{A}$*.*

Proof of Theorem 5.1: Let $\mathsf{DS}^* = \mathbf{JCl}[\mathsf{Sch}, \mathtt{CF}]$. By Theorem 4.1, we have

$$\mathbf{Adv}^{\mathrm{uf}}_{\overline{\mathsf{DS}},\mathbf{P}}(\mathcal{A}) \le \mathbf{Adv}^{\mathrm{prg}}_{\mathbf{P}_1}(\mathcal{A}_1) + \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{P}_2}(\mathcal{A}_2) + \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{P}_3}(\mathcal{A}_3) \ .$$

It is easy to see that

$$\mathbf{Adv}^{\mathrm{prg}}_{\mathbf{P}_1}(\mathcal{A}_1) \le \frac{\mathrm{q}^{\mathcal{A}_1}_{\mathrm{FO}}}{2^k} \le \frac{\mathrm{Q}^{\mathcal{A}}_{\mathrm{FO}}}{2^k}$$
$$\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{P}_2}(\mathcal{A}_2) \le \frac{\mathrm{q}^{\mathcal{A}_2}_{\mathrm{FO}}}{2^k} \le \frac{\mathrm{Q}^{\mathcal{A}}_{\mathrm{FO}}}{2^k} \ .$$

Under the assumption $\mathsf{p} > 2^{k-5}$ made in the theorem, BCJZ [15] established that $|\mathsf{Img}(\mathtt{CF})| = 2^{k-5}$. So $|\mathsf{Img}(\mathtt{CF})|/|\mathbb{Z}_{\mathsf{p}}| = 2^{k-5}/\mathsf{p} = \delta$. Let $\mathcal{B} = \mathcal{A}_3$ and note that $\mathbf{P}_3 = \mathbf{R}$. So by Theorem 4.2 we have

$$\mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{P}_3}(\mathcal{A}_3) \le (1/\delta) \cdot \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{R}}(\mathcal{B}) \ . \tag{4}$$

Collecting terms, we obtain the claimed bound stated in Theorem 5.1. ▌

<u>ANALYSIS OF THE **S** FUNCTOR.</u> Let $\overline{\mathsf{DS}}$ be the result of the **DtD** transform applied to Sch and a clamping function $\mathtt{CF} : \{0,1\}^k \to \mathbb{Z}_{\mathsf{p}}$. Security of EdDSA is captured as security in game $\mathbf{G}^{\mathrm{uf}}_{\overline{\mathsf{DS}},\mathbf{S}}$ when $\mathbf{S}$ is the functor that builds the component hash functions in the way that EdDSA does, namely from a MD-hash function. To evaluate this security, we start by defining the functor $\mathbf{S}$ in Figure 5. It is an arity-3 functor, and we separately specify $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3$. (Functor $\mathbf{S}_3$ will be called **Mod-MD** in later analyses.) The starting space, from which h is drawn, is $\mathsf{AF}(\{0,1\}^{b+2k}, \{0,1\}^{2k})$, the set of compression functions. The prime p is as before, and is public.

We want to establish the three assumptions of Theorem 4.1. Namely: (1) $\mathbf{S}_1$ is PRG-secure (2) $\mathbf{S}_2$ is PRF secure and (3) security holds in game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{Sch}^*,\mathbf{S}_3}$ where $\mathsf{Sch}^* = \mathbf{JCl}[\mathsf{Sch}, \mathtt{CF}]$. Bridging from Sch^* to Sch itself will use Theorem 4.2.

Lemma 5.2 *Let functor* $\mathbf{S}_1 : \mathsf{AF}(\{0,1\}^{b+2k}, \{0,1\}^{2k}) \to \mathsf{AF}(\{0,1\}^k, \{0,1\}^{2k})$ *be defined as in Figure 5. Let* $\mathcal{A}_1$ *be an adversary. Then*

$$\mathbf{Adv}^{\mathrm{prg}}_{\mathbf{S}_1}(\mathcal{A}_1) \le \frac{\mathrm{q}^{\mathcal{A}_1}_{\mathrm{FO}}}{2^k} \le \frac{\mathrm{Q}^{\mathcal{A}_1}_{\mathrm{FO}}}{2^k} \ . \tag{5}$$

Proof of Lemma 5.2: Since the input sk to $\mathbf{S}_1[\mathsf{h}]$ is k-bits long, the **MD** transform defined in Section 3 only iterates once and the output is $e = \mathsf{h}(IV \| sk \| P)$, for padding $P \in \{0,1\}^{3k}$ and

```
Games $\mathrm{G}_0$, $\boxed{\mathrm{G}_1}$

INIT:
  $sk \leftarrow\!\$ \{0,1\}^k$ ; $e \leftarrow\!\$ \{0,1\}^{2k}$
  Return $e$

$\mathrm{FO}(X)$:
  If $\mathrm{FT}[X] \neq \bot$ then return $\mathrm{FT}[X]$
  $Y \leftarrow\!\$ \{0,1\}^{2k}$
  If $X = IV \| sk \| P$ then $\mathsf{bad} \leftarrow \mathsf{true}$ ; $\boxed{Y \leftarrow e}$
  $\mathrm{FT}[X] \leftarrow Y$ ; Return $\mathrm{FT}[X]$

FIN$(c')$:
  Return $(c' = 1)$
```

Figure 6: Games G_0 and G_1 in the proof of Lemma 5.2. Boxed code is only in G_1.

initialization vector $IV \in \{0,1\}^{2k}$ that are fixed and known. Now consider the games in Figure 6, where the boxed code is only in G_1. Then we have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prg}}_{\mathbf{S}_1}(\mathcal{A}_1) &= \Pr[\mathrm{G}_1(\mathcal{A}_1)] - \Pr[\mathrm{G}_0(\mathcal{A}_1)] \\ &\leq \Pr[\mathrm{G}_0(\mathcal{A}_1) \text{ sets } \mathsf{bad}] \\ &\leq \frac{\mathrm{Q}^{\mathcal{A}_1}_{\mathrm{FO}}}{2^k} .\end{aligned}$$

The second line above is by the Fundamental Lemma of Game Playing, which applies since $\mathrm{G}_0, \mathrm{G}_1$ are identical-until-bad. ▌

We turn to PRF security of the $\mathbf{S}_2$ functor. Note that the construction is what BRT called AMAC [3]. They proved its PRF security by a combination of standard-model and ROM results. First they showed AMAC is PRF-secure if the compression function h is PRF-secure under leakage of a certain function of the key. Then they show that ideal compression functions have this PRF-under-leakage security. Putting this together implies PRF security of $\mathbf{S}_2$. However, we found it hard to put the steps and Lemmas in BRT together to get a good, concrete bound for the PRF security of $\mathbf{S}_2$. Instead we give a direct proof, with an explicit bound, using our result on the indifferentiability of **Mod-MD** from Theorem 6.1 together with the indifferentiability composition theorem [29].

Lemma 5.3 *Let functor* $\mathbf{S}_2 : \mathsf{AF}(\{0,1\}^{b+2k}, \{0,1\}^{2k}) \to \mathsf{AF}(\{0,1\}^k \times \{0,1\}^*, \mathbb{Z}_{\mathsf{p}})$ *be defined as in Figure 5. Let* ℓ *be an integer such that all messages queried to* FO *are no more than* $b \cdot (\ell - 1) - k$ *bits long. Let* $\mathcal{A}_2$ *be an adversary. Then*

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{S}_2}(\mathcal{A}_2) \leq \frac{\mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FO}}}{2^k} + \frac{2\mathsf{p}(\mathrm{q}^{\mathcal{A}_2}_{\mathrm{FO}} + \ell \mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FN}})}{2^{2k}} + \frac{(\mathrm{q}^{\mathcal{A}_2}_{\mathrm{FO}} + \ell \mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FN}})^2}{2^{2k}} + \frac{\mathsf{p}\mathrm{q}^{\mathcal{A}_2}_{\mathrm{FO}} \cdot \ell \mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FN}}}{2^{2k}} .$$

Proof of Lemma 5.3: In Section 6, we prove the indifferentiability of functor $\mathbf{S}_3$ (c.f. Figure 5), which we also call **Mod-MD**. Define $\mathbf{R} : \mathsf{AF}(\{0,1\}^*, \mathbb{Z}_{\mathsf{p}}) \to \mathsf{AF}(\{0,1\}^k \times \{0,1\}^*, \mathbb{Z}_{\mathsf{p}})$ to be the identity functor such that $\mathbf{R}[\mathsf{H}](x, y) = \mathsf{H}(x \,\|\, y)$ for all x, y, H in the appropriate domains. Notice that when $\mathbf{R}$ is given access to the **Mod-MD** functor as its oracle, the resulting functor is exactly $\mathbf{S}_2$. Using this property, we will reduce the PRF security of functor $\mathbf{S}_2$ to the indifferentiability of **Mod-MD**.

For any simulator algorithm $\mathcal{S}$, the indifferentiability composition theorem [29] grants the existence of distinguisher $\mathcal{D}$ and adversary $\mathcal{A}_5$ such that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{S}_2}(\mathcal{A}_2) \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{R}}(\mathcal{A}_5) + \mathbf{Adv}^{\mathrm{indiff}}_{\mathbf{Mod\text{-}MD},\mathcal{S}}(\mathcal{D}).$$

We let $\mathcal{S}$ be the simulator guaranteed by Theorem 6.1 and separately bound each of these terms. Adversary $\mathcal{A}_5$ simulates the PRF game for its challenger $\mathcal{A}_2$ by forwarding all FN queries to its own FN oracle and answering FO queries using the simulator, which has access to the FO oracle of $\mathcal{A}_5$. Since the simulator is efficient and makes at most one query to its oracle each time it is run, we can say the runtime of $\mathcal{A}_5$ is approximately the same as that of $\mathcal{A}_2$. $\mathcal{A}_5$ makes the same number of FN and FO queries as $\mathcal{A}_2$.

Next, we want to compute $\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{R}}(\mathcal{A}_5)$. When $\mathbf{R}$ is evaluated with access to a random function h, its outputs are random unless the adversary makes a relevant query involving the secret key. The adversary can only distinguish if the output of FN is randomly sampled or from $\mathbf{R}[\mathsf{h}]$ if it queries FO on the k-bit secret key (e_2), which has probability $\frac{1}{2^k}$ for a single query. Taking a union bound over all FO queries, we have

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{R}}(\mathcal{A}_5) \leq \frac{\mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FO}}}{2^k}.$$

Distinguisher $\mathcal{D}$ simulates the PRF game for $\mathcal{A}_2$, by replacing functor **Mod-MD** with its own Priv oracle within the FN oracle and forwarding $\mathcal{A}_2$'s direct FO queries to Pub. $\mathcal{D}$ hence makes $\mathrm{Q}^{\mathrm{FN}}_{\mathcal{A}_2}$ queries to Priv of maximum length $b\cdot(\ell-1)$ and $\mathrm{q}^{\mathrm{FO}}_{\mathcal{A}_2}$ to Pub. To bound the second term, we apply Theorem 6.1 on the indifferentiability of shrink-MD transforms. This theorem is parameterized by two numbers γ and ϵ; in Section 6, we show that **Mod-MD** belongs to the shrink-MD class for $\gamma = \lfloor \frac{2^{2k}}{\mathsf{p}} \rfloor$ and $\epsilon = \frac{\mathsf{p}}{2^{2k}}$. Then the theorem gives

$$\mathbf{Adv}^{\mathrm{indiff}}_{\mathbf{Mod\text{-}MD},\mathcal{S}}(\mathcal{D}) \leq 2(\mathrm{Q}^{\mathcal{D}}_{\textsc{Pub}} + \ell\mathrm{Q}^{\mathcal{D}}_{\textsc{Priv}})\epsilon + \frac{(\mathrm{Q}^{\mathcal{D}}_{\textsc{Pub}} + \ell\mathrm{Q}^{\mathcal{D}}_{\textsc{Priv}})^2}{2^{2k}} + \frac{\mathrm{Q}^{\mathcal{D}}_{\textsc{Pub}}\cdot\ell\mathrm{Q}^{\mathcal{D}}_{\textsc{Priv}}}{\gamma}.$$

By substituting $\mathrm{Q}^{\mathcal{D}}_{\textsc{Pub}} = \mathrm{q}^{\mathcal{A}_2}_{\mathrm{FO}}$ and $\mathrm{Q}^{\mathcal{D}}_{\textsc{Priv}} = \mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FN}}$, we obtain the bound stated in the theorem.▮

Finally we turn to $\mathbf{S}_3$. The following considers the UF security of $\mathsf{DS}^* = \mathbf{JCl}[\mathsf{Sch},\mathsf{CF}]$ with the hash function being an MD one, meaning with $\mathbf{S}_3$, and reduces this to the UF security of the same scheme with the hash function being a monolithic random oracle. Formally, the latter is captured by game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{R}}$ where $\mathbf{R}$ is the identity functor. One route to this result is to exploit the public-indifferentiability of **MD** established by DRS [20]. However we found it simpler to give a direct proof and bound based on our Theorem 6.1.

Lemma 5.4 *Let functor $\mathbf{S}_3 : \mathsf{AF}(\{0,1\}^{b+2k},\{0,1\}^{2k}) \to \mathsf{AF}(\{0,1\}^*,\mathbb{Z}_{\mathsf{p}})$ be defined as in Figure 5. Assume $2^k > \mathsf{p}$. Let $\mathsf{DS}^* = \mathbf{JCl}[\mathsf{Sch},\mathsf{CF}]$ where $\mathsf{CF}: \{0,1\}^k \to \mathbb{Z}_p$ is a clamping function. Let $\mathbf{R} : \mathsf{AF}(\{0,1\}^*,\mathbb{Z}_{\mathsf{p}}) \to \mathsf{AF}(\{0,1\}^*,\mathbb{Z}_{\mathsf{p}})$ be the identity functor, meaning $\mathbf{R}[\mathsf{H}] = \mathsf{H}$. Let $\mathcal{A}_3$ be a $\mathbf{G}^{\mathrm{uf}}$ adversary and let ℓ be an integer such that the maximum message length $\mathcal{A}_3$ queries to* Sign *is at most $b\cdot(\ell-1)-2k$ bits. Then we can construct adversary $\mathcal{A}_4$ such that*

$$\mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{S}_3}(\mathcal{A}_3) \leq \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{R}}(\mathcal{A}_4) + \frac{2\mathsf{p}(\mathrm{q}^{\mathcal{A}_3}_{\mathrm{FO}} + \ell\mathrm{Q}^{\mathcal{A}_3}_{\textsc{Sign}})}{2^{2k}} \tag{6}$$

$$+ \frac{(\mathrm{q}^{\mathcal{A}_3}_{\mathrm{FO}} + \ell\mathrm{Q}^{\mathcal{A}_3}_{\textsc{Sign}})^2}{2^{2k}} + \frac{\mathsf{p}\mathrm{q}^{\mathcal{A}_3}_{\mathrm{FO}}\cdot\ell\mathrm{Q}^{\mathcal{A}_3}_{\textsc{Sign}}}{2^{2k}}\,. \tag{7}$$

Adversary $\mathcal{A}_4$ has approximately equal runtime and query complexity to $\mathcal{A}_3$.

Proof of Lemma 5.4: Again, we rely on the indifferentiability of functor $\mathbf{S}_3 = \mathbf{Mod\text{-}MD}$, as shown in Section 6. The general indifferentiability composition theorem [29] states that for any simulator $\mathcal{S}$ and adversary $\mathcal{A}_3$, there exist distinguisher $\mathcal{D}$ and adversary $\mathcal{A}_4$ such that

$$\mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{S}_3}(\mathcal{A}_3) \leq \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{R}}(\mathcal{A}_4) + \mathbf{Adv}^{\mathrm{indiff}}_{\mathbf{S}_3,\mathcal{S}}(\mathcal{D}).$$

Let $\mathcal{S}$ be the simulator whose existence is implied by Theorem 6.1. The distinguisher runs the unforgeability game for its adversary, replacing $\mathbf{S}_3[\mathrm{FO}]$ in scheme algorithms and adversarial FO queries with its Priv and Pub oracles respectively. It makes $\mathrm{q}^{\mathcal{A}_3}_{\mathrm{FO}}$ queries to Pub and $\mathrm{Q}^{\mathcal{A}_3}_{\textsc{Sign}}$ queries to Priv, and the maximum length of any query to Priv is $b \cdot (\ell - 1)$ bits because each element of group $\mathbb{G}_{\mathsf{p}}$ is a k-bit string (c.f. Section 2). We apply Theorem 6.1 to obtain the bound

$$\mathbf{Adv}^{\mathrm{indiff}}_{\mathbf{S}_3,\mathcal{S}}(\mathcal{D}) \leq 2(\mathrm{q}^{\mathcal{A}_3}_{\mathrm{FO}} + \ell \mathrm{Q}^{\mathcal{A}_3}_{\textsc{Sign}})\epsilon + \frac{(\mathrm{q}^{\mathcal{A}_3}_{\mathrm{FO}} + \ell \mathrm{Q}^{\mathcal{A}_3}_{\textsc{Sign}})^2}{2^{2k}} + \frac{\mathrm{q}^{\mathcal{A}_3}_{\mathrm{FO}} \cdot \ell \mathrm{Q}^{\mathcal{A}_3}_{\textsc{Sign}}}{\gamma}.$$

Adversary $\mathcal{A}_4$ is a wrapper for $\mathcal{A}_3$, which answers all of its queries to FO by running $\mathcal{S}$ with access to its own FO oracle; since the simulator runs in constant time and makes only one query to its oracle, the runtime and query complexity approximately equal those of $\mathcal{A}_3$.

Substituting $\frac{1}{\gamma} \geq \frac{\mathsf{p}}{2^{2k}}$ and $\epsilon = \frac{\mathsf{p}}{2^{2k}}$ gives the bound. ▌

<u>Security of EdDSA with MD.</u> We now want to conclude security of EdDSA, with an MD-hash function, assuming security of $\mathsf{Schnorr}$ with a monolithic random oracle. The Theorem is for a general prime p in the range $2^k > \mathsf{p} > 2^{k-5}$ but in EdDSA the prime is $2^{k-4} < \mathsf{p} < 2^{k-3}$ so the value of δ below is $\delta = 2^{k-5}/\mathsf{p} > 2^{k-5}/2^{k-3} = 1/4$, so the factor $1/\delta$ is ≤ 4. Again recall our convention that query counts of an adversary include those made by oracles in its game, implying for example that $\mathrm{Q}^{\mathcal{A}}_{\mathrm{FO}} \geq \mathrm{Q}^{\mathcal{A}}_{\textsc{Sign}}$.

Theorem 5.5 *Let* $\mathsf{DS} = \mathsf{Sch}$ *be the* $\mathsf{Schnorr}$ *signature scheme of Figure 4. Let* $\mathtt{CF}\colon \{0,1\}^k \to \mathbb{Z}_{\mathsf{p}}$ *be the clamping function of Figure 4. Assume* $2^k > \mathsf{p} > 2^{k-5}$ *and let* $\delta = 2^{k-5}/\mathsf{p}$. *Let* $\overline{\mathsf{DS}} = \mathbf{DtD}[\mathsf{DS}, \mathtt{CF}]$ *be the* EdDSA *signature scheme. Let* $\mathbf{R}\colon \mathsf{AF}(\{0,1\}^*, \mathbb{Z}_{\mathsf{p}}) \to \mathsf{AF}(\{0,1\}^*, \mathbb{Z}_{\mathsf{p}})$ *be the identity functor. Let* $\mathbf{S}$ *be the functor of Figure 5. Let* $\mathcal{A}$ *be an adversary attacking the* $\mathbf{G}^{\mathrm{uf}}$ *security of* $\overline{\mathsf{DS}}$. *Again let* $b \cdot (\ell - 1) - 2k$ *be the maximum length in bits of a message input to* Sign. *Then there is an adversary* $\mathcal{B}$ *such that*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{uf}}_{\overline{\mathsf{DS}},\mathbf{S}}(\mathcal{A}) \leq & (1/\delta) \cdot \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{R}}(\mathcal{B}) + \frac{\mathrm{Q}^{\mathcal{A}}_{\mathrm{FO}}}{2^{k-1}} + \frac{\mathsf{p}(\mathrm{q}^{\mathcal{A}}_{\mathrm{FO}} + \ell \mathrm{Q}^{\mathcal{A}}_{\textsc{Sign}})}{2^{2k-2}} \\ & + \frac{(\mathrm{q}^{\mathcal{A}}_{\mathrm{FO}} + \ell \mathrm{Q}^{\mathcal{A}_2}_{\textsc{Sign}})^2}{2^{2k-1}} + \frac{\mathsf{p}\mathrm{q}^{\mathcal{A}}_{\mathrm{FO}} \cdot \ell \mathrm{Q}^{\mathcal{A}}_{\textsc{Sign}}}{2^{2k-1}} \,.\end{aligned}$$

Adversary $\mathcal{B}$ *preserves the queries and running time of* $\mathcal{A}$.

Proof of Theorem 5.5: Let $\mathsf{DS}^* = \mathbf{JCl}[\mathsf{Sch}, \mathtt{CF}]$. By Theorem 4.1, we have

$$\mathbf{Adv}^{\mathrm{uf}}_{\overline{\mathsf{DS}},\mathbf{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prg}}_{\mathbf{S}_1}(\mathcal{A}_1) + \mathbf{Adv}^{\mathrm{prf}}_{\mathbf{S}_2}(\mathcal{A}_2) + \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{S}_3}(\mathcal{A}_3).$$

Now applying Lemma 5.2, we have

$$\mathbf{Adv}^{\mathrm{prg}}_{\mathbf{S}_1}(\mathcal{A}_1) \leq \frac{\mathrm{Q}^{\mathcal{A}}_{\mathrm{FO}}}{2^k} \,.$$

Applying Lemma 5.3, we have

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathbf{S}_2}(\mathcal{A}_2) \leq \frac{\mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FO}}}{2^k} + \frac{2\mathsf{p}(\mathrm{q}^{\mathcal{A}_2}_{\mathrm{FO}} + \ell \mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FN}})}{2^{2k}} + \frac{(\mathrm{q}^{\mathcal{A}_2}_{\mathrm{FO}} + \ell \mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FN}})^2}{2^{2k}} + \frac{\mathsf{p}\mathrm{q}^{\mathcal{A}_2}_{\mathrm{FO}} \cdot \ell \mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FN}}}{2^{2k}}.$$

We substitute $\mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FO}} = \mathrm{Q}^{\mathcal{A}}_{\mathrm{FO}}$, $\mathrm{q}^{\mathcal{A}_2}_{\mathrm{FO}} = \mathrm{q}^{\mathcal{A}}_{\mathrm{FO}}$ and $\mathrm{Q}^{\mathcal{A}_2}_{\mathrm{FN}} = \mathrm{Q}^{\mathcal{A}}_{\textsc{Sign}}$. By Lemma 5.4 we obtain

$$\begin{aligned}\mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{S}_3}(\mathcal{A}_3) \leq & \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{R}}(\mathcal{B}) + \frac{2\mathsf{p}(\mathrm{Q}^{\mathcal{A}_3}_{\mathrm{FO}} + \ell \mathrm{Q}^{\mathcal{A}_3}_{\textsc{Sign}})}{2^{2k}} \\ & + \frac{(\mathrm{Q}^{\mathcal{A}_3}_{\mathrm{FO}} + \ell \mathrm{Q}^{\mathcal{A}_3}_{\textsc{Sign}})^2}{2^{2k}} + \frac{\mathsf{p}\mathrm{Q}^{\mathcal{A}_3}_{\mathrm{FO}} \cdot \ell \mathrm{Q}^{\mathcal{A}_3}_{\textsc{Sign}}}{2^{2k}} \,.\end{aligned}$$

```
Game $\mathbf{G}^{\mathrm{indiff}}_{\mathbf{F},\mathcal{S}}$

INIT():
  $c \leftarrow\!\!\$\ \{0,1\}$
  $\mathsf{h} \leftarrow\!\!\$\ \mathsf{SS}$
  $\mathsf{H} \leftarrow\!\!\$\ \mathsf{ES}$

PUB($i, Y$):
  if $c = 0$ then
      return $\mathcal{S}[\mathsf{H}](i, Y)$
  else return $\mathsf{h}(i, Y)$

PRIV($i, X$):
  if $c = 0$ then return $\mathsf{H}(i, X)$
  else return $\mathbf{F}[\mathsf{h}](i, X)$

FIN($c'$):
  return $[[c = c']]$
```

Figure 7: The game $\mathbf{G}^{\mathrm{indiff}}_{\mathbf{F},\mathcal{S}}$ measuring indifferentiability of a functor $\mathbf{F}$ with respect to simulator $\mathcal{S}$.

Recall that adversary $\mathcal{A}_3$ has the same query complexity as $\mathcal{A}$.

Under the assumption $\mathsf{p} > 2^{k-5}$ made in the theorem, BCJZ [15] established that $|\mathsf{Img}(\mathtt{CF})| = 2^{k-5}$. So $|\mathsf{Img}(\mathtt{CF})|/|\mathbb{Z}_{\mathsf{p}}| = 2^{k-5}/\mathsf{p} = \delta$. So by Theorem 4.2 we have

$$\mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}^*,\mathbf{R}}(\mathcal{B}) \le (1/\delta) \cdot \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{R}}(\mathcal{B}) \ . \tag{8}$$

By substituting with the number of queries made by $\mathcal{A}$ as in Theorem 4.1 and collecting terms, we obtain the claimed bound stated in Theorem 5.5. ▌

We can now obtain security of EdDSA under number-theoretic assumptions via known results on the security of $\mathsf{Schnorr}$. Namely, we use the known results to bound $\mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS},\mathbf{R}}(\mathcal{B})$ above. From [36, 1] we can get a bound and proof based on the DL problems, and from [38] with a better bound. We can also get an almost tight bound under the MBDL assumption via [5] and a tight bound in the AGM via [21].

6 Indifferentiability of the shrink-MD class of functors

INDIFFERENTIABILITY We want the tuple of functions returned by a functor $\mathbf{F} : \mathsf{SS} \to \mathsf{ES}$ to be able to "replace" a tuple drawn directly from ES. Indifferentiability is a way of defining what this means. We adapt the original MRH definition of indifferentiability [29] to our game-based model in Figure 7. In this game, $\mathcal{S}$ is a simulator algorithm. The advantage of an adversary $\mathcal{A}$ against the indifferentiability of functor $\mathbf{F}$ with respect to simulator $\mathcal{S}$ is defined to be

$$\mathbf{Adv}^{\mathrm{indiff}}_{\mathbf{F},\mathcal{S}}(\mathcal{A}) := 2\Pr[\mathbf{G}^{\mathrm{indiff}}_{\mathbf{F},\mathcal{S}}(\mathcal{A}) \Rightarrow 1] - 1.$$

MODIFYING THE MERKLE-DAMGÅRD TRANSFORM Coron et al. showed that the Merkle-Damgård transform is not indifferentiable with respect to any efficient simulator due to its susceptibility to length-extension attacks [18]. In the same work, they analysed the indifferentiability of several closely related indifferentiable constructions, including the "chop-MD" construction. Chop-MD is a functor with the same domain as the MD transform; it simply truncates a specified number of bits from the output of MD. The $\mathbf{S}_3$ functor of Figure 5 operates similarly to the chop-MD functor, except that $\mathbf{S}_3$ reduces the output modulo a prime p instead of truncating. This small change introduces some bias into the resulting construction that affects its indifferentiability due to the fact that the outputs of the MD transform, which are $2k$-bit strings, are not distributed uniformly over $\mathbb{Z}_{\mathsf{p}}$.

In this section, we establish indifferentiability for a general class of functors that includes both chop-MD and $\mathbf{S}_3$. We rely on the indifferentiability of $\mathbf{S}_3$ in Section 5 as a stepping-stone to the unforgeability of EdDSA; however, we think our proof for chop-MD is of independent interest and improves upon prior work.

The original analysis of the chop-MD construction [18] was set in the ideal cipher model and accounted for some of the structure of the underlying compression function. A later proof by Fischlin and Mittelbach [31] adapts the proof strategy to the simpler construction we address here and works in the random oracle model as we do. Both proofs, however, contain a subtle gap in the way they use their simulators.

At a high level, both proofs define stateful simulators $\mathcal{S}$ which simulate a random compression function by sampling uniform answers to some queries and programming others with the help of their random oracles. These simulators are not perfect, and fail with some probability that the proofs bound. In the ideal indifferentiability game, the PUB oracle answers queries using the simulator and the PRIV oracle answers queries using a random oracle. Both proofs at some point replace the random oracle H in PRIV with $\mathbf{Chop\text{-}MD}[\mathcal{S}]$ and claim that because $\mathbf{Chop\text{-}MD}[\mathcal{S}[\mathsf{H}]](X)$ will always return $\mathsf{H}(X)$ if the simulator does not fail, the adversary cannot detect the change. This argument is not quite true, because the additional queries to $\mathcal{S}$ made by the PRIV oracle can affect its internal state and prevent the simulator from failing when it would have in the previous game. In our proof, we avoid this issue with a novel simulator with *two internal states* to enforce separation between PRIV and PUB queries that both run the simulator.

Our result establishes indifferentiability for all members of the **Shrink-MD** class of functors, which includes any functor built by composing of the MD transform with a function $\mathsf{Out} : \{0,1\}^{2k} \to S$ that satisfies three conditions, namely that for some $\gamma, \epsilon \geq 0$,

1. For all $y \in S$, we can efficiently sample from the uniform distribution on the preimage set $\{\mathsf{Out}^{-1}(y)\}$. We permit the sampling algorithm to fail with probability at most ϵ, but require that upon failure the algorithm outputs a (not necessarily random) element of $\{\mathsf{Out}^{-1}(y)\}$.

2. For all $y \in S$, it holds that $\gamma \leq |\{\mathsf{Out}^{-1}(y)\}|$.

3. The statistical distance $\delta(D)$ between the distribution
$$D := z \leftarrow\!\!\$\ \mathsf{Out}^{-1}(y) \colon y \leftarrow\!\!\$\ S$$
and the uniform distribution on $\{0,1\}^{2k}$ is bounded above by ϵ.

In principle, we wish γ to be large and ϵ to be small; if this is so, then the set S will be substantially smaller than $\{0,1\}^{2k}$ and the function Out "shrinks" its domain by mapping it onto a smaller set.

Both chop-MD and mod-MD are members of the **Shrink-MD** class of functors; we briefly show the functions that perform bit truncation and modular reduction by a prime satisfy our three conditions. Truncation by any number of bits trivially satisfies condition (1) with $\epsilon = 0$.

Reduction modulo p also satisfies condition (1) because the following algorithm samples from the equivalence class of x modulo p with failure probability at most $\frac{\mathsf{p}}{2^{2k}}$. Let ℓ be the smallest integer such that $\ell > \frac{2^{2k}}{\mathsf{p}}$. Sample $w \leftarrow\!\!\$\ [0 \ldots \ell - 1]$ and output $w \cdot \mathsf{p} + x$, or x if $w \cdot \mathsf{p} + x > 2^{2k}$. We say this algorithm "fails" in the latter case, which occurs with probability at most $\frac{1}{\ell} < \frac{\mathsf{p}}{2^{2k}}$. In the event the algorithm does not fail, it outputs a uniform element of the equivalence class of x.

Bellare et al. showed that the truncation of n trailing bits satisfies condition (2) for $\gamma = 2^{2k-n}$ and reduction modulo prime p satisfies (2) for $\gamma = \lfloor 2^{2k}/\mathsf{p} \rfloor$. It is clear that sampling from the preimages of a random $2k - n$-bit string under n-bit truncation produces a uniform $2k$-bit string, so truncation satisfies condition (3) with $\epsilon = 0$. Also from Bellare et al. [3], we have that the statistical distance between a uniform element of $\mathbb{Z}_{\mathsf{p}}$ and the modular reduction of a uniform $2k$-bit string is $\epsilon = \frac{\mathsf{p}}{2^{2k}}$. The statistical distance of our distribution $z \leftarrow\!\!\$\ \mathsf{Out}^{-1}(Y)$ for uniform Y over S from the uniform distribution over $\{0,1\}^{2k}$ is bounded above by the same ϵ; hence condition (3) holds.

Given a set S and a function $\mathsf{Out} : \{0,1\}^{2k} \to S$, we define the functor $\mathbf{F}_{S,\mathsf{Out}}$ as the composition of Out with $\mathbf{MD}$. In other words, for any $x \in \{0,1\}^*$ and $\mathsf{h} \in \mathsf{AF}(\{0,1\}^{b+2k}, \{0,1\}^{2k})$, let $\mathbf{F}_{S,\mathsf{Out}}[\mathsf{h}](x) := \mathsf{Out}(\mathbf{MD}[\mathsf{h}](x))$.

Theorem 6.1 *Let k be an integer and S a set of bitstrings. Let $\mathsf{Out} : \{0,1\}^{2k} \to S$ be a function satisfying conditions (1), (2), and (3) above with respect to $\gamma, \epsilon > 0$. Let $\mathbf{MD}$ be the Merkle-Damgård functor(c.f. Section 2) $\mathbf{F}_{S,\mathsf{Out}} := \mathsf{Out}\circ\mathbf{MD}$ be the functor described in the prior paragraph. Let pad be the padding function used by $\mathbf{MD}$, and let unpad be the function that removes padding from its input (i.e., for all $X \in \{0,1\}^*$, it holds that $\mathsf{unpad}(X \,\|\, \mathsf{pad}(|X|)) = X$). Assume that unpad returns $\perp$ if its input is incorrectly padded and that unpad is injective on its support. Then there exists a simulator $\mathcal{S}$ such that for any adversary $\mathcal{A}$ making* Priv *queries of maximum length $b \cdot (\ell - 1)$ bits then*

$$\mathbf{Adv}^{\mathsf{indiff}}_{\mathbf{F},\mathcal{S}}(\mathcal{A}) \le 2(\mathrm{Q}^{\mathcal{A}}_{\mathrm{PUB}} + \ell\mathrm{Q}^{\mathcal{A}}_{\mathrm{PRIV}})\epsilon + \frac{(\mathrm{Q}^{\mathcal{A}}_{\mathrm{PUB}} + \ell\mathrm{Q}^{\mathcal{A}}_{\mathrm{PRIV}})^2}{2^{2k}} + \frac{\mathrm{Q}^{\mathcal{A}}_{\mathrm{PUB}} \cdot \ell\mathrm{Q}^{\mathcal{A}}_{\mathrm{PRIV}}}{\gamma}.$$

Proof of Theorem 6.1: We first give a brief overview of our proof strategy and its differences from previous indifferentiability proofs for the chop-MD construction [18, 31].

Our simulator, $\mathcal{S}$, is defined in Figure 8. It is inspired by, but distinct from, that of Mittelbach and Fischlin's simulator for the chop-MD construction ([31] Figure 17.4.), which in turn adapts the simulator of Coron et al [18] from the ideal cipher model to the random oracle model. These simulators all present the interface of a random compression function h and internally maintain a graph in which each edge represents an input-output pair under the simulated compression function. The intention is that each path through this graph will represent a possible evaluation of $\mathbf{F}_{S,\mathsf{Out}}[\mathsf{h}]$. The fundamental difference between our simulator and previous ones is that we maintain two internal graphs instead of one: one graph for all queries, and one graph for public interface queries only. This novel method of using two graphs avoids the gap in prior proofs described above by tracking precisely which parts of the simulator's state are influenced by private and public interface queries respectively.

In the "ideal" indifferentiability game, Priv queries are answered by random oracle $\mathsf{H} \leftarrow\!\!\$\ \mathsf{AF}\{0,1\}^*, S$. Pub queries are answered by the simulator $\mathcal{S}$, which maintains the two graphs $\mathcal{G}_{\mathtt{pub}}$ and $\mathcal{G}_{\mathtt{all}}$. We present pseudocode for this game (G_0) in Figure 8. In each graph, the nodes and edges are labeled with $2k$-bit strings. An edge from node y to node z with label m is denoted (y, z, m), and represents a single value of the simulated compression function; namely, on $6k$-bit input $y \,\|\, m$, the simulated compression function should output z. Queries made in the process of evaluating $\mathbf{MD}[S]$ will form a path that begins at the node labeled with the initialization vector IV; the path's edges will be labeled with the $4k$-bit blocks of $\mathsf{pad}(M)$.

Whenever the simulator receives a fresh query (y, m), it uses a pathfinding algorithm FindPath to check whether the query extends an existing path from IV and thus continues an existing evaluation of the MD transform. If so, it reads the message from the path's edge labels then appends the new block m to the end. If the result is a properly padded message, the simulator removes the padding and uses its oracle H to compute the output of functor $\mathbf{F}$ on the original message. This output w is an element of S, and it should be consistent with Out when applied to the $2k$-bit simulator output. The simulator therefore samples its response from the preimages of w under Out. If any of these steps fail, then the query does not need to be programmed, so the simulator samples a uniformly

```
Simulator $\mathcal{S}[\mathsf{H}](Y, \mathcal{G})$ :

1  $(y, m) \leftarrow Y$
2  if $\exists z$ such that $(y, z, m) \in \mathcal{G}$.edges
3     return $z$
4  $M \leftarrow \mathcal{G}$.FindPath$(IV, y)$
5  if $M \neq \bot$ and $\mathsf{unpad}(M \,\|\, m) \neq \bot$ then
6     if $\mathrm{T}_{\mathsf{h}}[Y, M] \neq \bot$ then $z \leftarrow \mathrm{T}_{\mathsf{h}}[Y, M]$
7     else $z \leftarrow\!\!\$\ \mathsf{Out}^{-1}(\mathsf{H}(\mathsf{unpad}(M \,\|\, m)))$
8        $\mathrm{T}_{\mathsf{h}}[Y, M] \leftarrow z$
9  else if $\mathrm{T}_{\mathsf{h}}[Y] \neq \bot$ then $z \leftarrow \mathrm{T}_{\mathsf{h}}[Y]$
10 else $z \leftarrow\!\!\$\ \{0,1\}^{2k}$; $\mathrm{T}_{\mathsf{h}}[Y] \leftarrow z$
11 add $(y, z, m)$ to $\mathcal{G}$.edges
12 add $(y, z, m)$ to $\mathcal{G}_{\texttt{all}}$.edges
13 return $z$
```

```
Game $\mathrm{G}_0 := \mathbf{G}^{\mathrm{indiff}}_{\mathbf{F},\mathcal{S}} | b = 0$

INIT():
1  $\mathsf{H} \leftarrow\!\!\$\ \mathsf{AF}\{0,1\}^*, S$
2  $\mathcal{G}_{\texttt{all}}, \mathcal{G}_{\texttt{pub}} \leftarrow (IV)$

PRIV(X):
1  return $\mathsf{H}(X)$

PUB(Y):
1  $z \leftarrow \mathcal{S}[\mathsf{H}](Y, \mathcal{G}_{\texttt{pub}})$
2  return $z$

FIN(c'):
1  return $c'$
```

Figure 8: Left: Indifferentiability simulator for the proof of Theorem 6.1. Right: The ideal game $\mathbf{G}^{\mathrm{indiff}}_{\mathbf{F},\mathcal{S}}$ measuring indifferentiability of a functor $\mathbf{F}$ with respect to simulator $\mathcal{S}$

random response z and updates its graph with the new edge from y. Because we are attempting to simulate a random function, the simulator must cache its responses to maintain consistency between repeated queries. It does this in two ways: via the graphs and via table T_{h}. We require two forms of caching because the simulator may use two graphs and thus responses may not be cached consistently between private and public queries in the graphs alone.

Our G_0 differs from this ideal indifferentiability game only in the FIN oracle, which returns the adversary's challenge guess c'. Thus the probability that game G_0 returns 1 exactly equals $1 - \Pr[\mathbf{G}^{\mathrm{indiff}}_{\mathbf{F},\mathcal{S}}(\mathcal{A}) | c = 0]$.

We move to G_1, where the PRIV oracle uses $\mathcal{S}$ to calculate the output of functor $\mathbf{F}$, then discards the result. We wish for the adversary's view of games G_0 and G_1 to be identical, so we must ensure that the additional queries to $\mathcal{S}$ do not influence its state or its responses to PUB queries. We therefore call the simulator with different graphs in the two oracles. It responds to public queries based only on the public graph, and queries made by PRIV are private and do not update the public graph. We do use shared table T_{h} to cache outputs across all queries; in this sense a private query can affect a public query; however, we cache responses separately for each branch of the simulator, so our caching does not alter the simulator's branching behavior and the distribution of public queries' responses does not change. The adversary cannot detect at what time a response z is first sampled, so its view does not change, and

$$\Pr[\mathrm{G}_0] = \Pr[\mathrm{G}_1].$$

In game G_2, we set a bad flag if the simulator if $\mathcal{G}_{\texttt{all}}$ contains any collisions, cycles, or "duplicate" edges: edges with the same starting node and label but different ending nodes.

Collisions and cycles are formed only when a new edge is created whose ending node is already present in the graph; we set bad in this case. The caching in line 2 prevents duplicate edges except when the PRIV and PUB oracles query the simulator on the same input (y, m), in that order. Even in this case, caching in table T_{h} prevents duplicate edges unless one query detects a path that the

```
Game G_1

INIT():
  $\mathsf{H} \leftarrow_\$ \mathsf{AF}\{0,1\}^*, S$
  $\mathcal{G}_{\mathtt{all}}, \mathcal{G}_{\mathtt{pub}} \leftarrow (IV)$

PUB(Y):
  $z \leftarrow \mathcal{S}[\mathsf{H}](Y, \mathcal{G}_{\mathtt{pub}})$
  return $z$

PRIV(X):
  $w \leftarrow \mathbf{F}[\mathcal{S}[\mathsf{H}](\cdot, \mathcal{G}_{\mathtt{all}})](X)$
  return $\mathsf{H}(X)$

FIN(c'):
  return $c'$
```

Figure 9: Game G_1 in the proof of Theorem 6.1. Highlighted code is changed from the previous game, and algorithms not shown are unchanged from the previous game.

other did not, or the two queries detect different paths.

If the PUB query detects a path to node y that did not exist during the previous PRIV query, or there are two distinct paths to y in $\mathcal{G}_{\mathtt{all}}$, then $\mathcal{G}_{\mathtt{all}}$ must contain a collision or a cycle, and the bad flag will be set when that is detected. Furthermore, $\mathcal{G}_{\mathtt{pub}}$ is a subgraph of $\mathcal{G}_{\mathtt{all}}$, so it cannot contain a path to y that $\mathcal{G}_{\mathtt{all}}$ does not. To catch the formation of duplicate edges, it is therefore sufficient to set bad if $\mathcal{G}_{\mathtt{all}}$ contains a path from IV to y that is not detected by the subsequent PUB query.

The bad flag is internal and does not affect the view of the game, so

$$\Pr[\mathrm{G}_2] = \Pr[\mathrm{G}_1]$$

In G_3, we force the adversary to lose when the bad flag is set. This strictly decreases their advantage, so

$$\Pr[\mathrm{G}_3] \leq \Pr[\mathrm{G}_2].$$

In our next game, we stop querying H directly in the PRIV oracle and instead return w, the result of our functor on the query. We claim that in G_3, either $w = \mathsf{H}(X)$ or $\mathsf{bad} = \mathsf{true}$; thus if the adversary wins G_3, then in all PRIV queries we have $w = \mathsf{H}(X)$. From this claim, we can see that the change does not affect the view of the adversary and

$$\Pr[\mathrm{G}_4] = \Pr[\mathrm{G}_3].$$

To prove the claim, consider a query PRIV(X). Let $(X_1, \ldots, X_n)$ be the b-bit blocks of $X \,\|\, \mathsf{pad}(|X|)$. By the definition of the MD transform, PRIV makes n queries to $\mathcal{S}$ of the form $(y_i, X_i), \mathcal{G}_{\mathtt{all}}$, where $y_1 = IV$ and $y_i = \mathcal{S}((y_{i-1}, X_{i-1}), \mathcal{G}_{\mathtt{all}})$ for all $i > 1$. These may not be fresh queries, but they must be made in order or bad will be set: if query $\mathcal{S}((y_i, X_i))$ outputs y_{i+1} and this has already been the input of a prior query, then y_{i+1} is a node in $\mathcal{G}_{\mathtt{all}}$; a collision has occurred and the query will set bad. Unless bad is set, there exists exactly one path in $\mathcal{G}_{\mathtt{all}}$ from IV to y_i, and the labels on this path are $(X_1, \ldots, X_{i-1})$. This is trivially true for $i = 1$; the path is the empty path. The query $\mathcal{S}((y_{i-1}, X_{i-1}), \mathcal{G}_{\mathtt{all}})$ creates the edge (y_{i-1}, y_i, X_{i-1}) in $\mathcal{G}_{\mathtt{all}}$. By induction on i, there is always a path from IV to y_i with labels $(X_1, \ldots, X_{i-1})$. If there exists more than one path from IV to y_i, then $\mathcal{G}_{\mathtt{all}}$ must contain either a cycle or two edges with the same ending node; in either case the bad flag will be set.

Therefore, when PRIV first makes the query $\mathcal{S}((y_{n-1}, X_n), \mathcal{G}_{all})$, it will detect the path, compute $\mathsf{unpad}(M \,\|\, X_n) = X$ and output an element $z \in \mathsf{Out}^{-1}(\mathsf{H}(X))$. By the definition of Out^{-1}, we have $w = \mathsf{Out}(z) = \mathsf{H}(X)$, so the claim holds.

```
Game $\mathrm{G}_2$, $\boxed{\mathrm{G}_3}$

$\textsc{Fin}(c')$:
1  $\boxed{\text{if bad then return } 0}$
2  return $c'$
```

```
Game $\mathrm{G}_4$

$\textsc{Priv}(X)$:
1  $w \leftarrow \mathbf{F}[\mathcal{S}[\mathsf{H}](\cdot, \mathcal{G}_{\mathtt{all}})](X)$
2  return $w$
```

```
$\mathcal{S}[\mathsf{H}](Y, \mathcal{G})$:
1   $(y, m) \leftarrow Y$
2   if $\exists z$ such that $(y, z, m) \in \mathcal{G}$.edges
3     return $z$
4   $M \leftarrow \mathcal{G}$.FindPath$(IV, y)$
5   $M_{\mathtt{all}} \leftarrow \mathcal{G}_{\mathtt{all}}$.FindPath$(IV, y)$
6   if $M \neq \bot$ and $\mathsf{unpad}(M \,\|\, m) \neq \bot$ then
7     if $\mathrm{T}_{\mathsf{h}}[Y, M] \neq \bot$ then $z \leftarrow \mathrm{T}_{\mathsf{h}}[Y, M]$
8     else $z \leftarrow\!\!\$\; \mathsf{Out}^{-1}(\mathsf{H}(\mathsf{unpad}(M \,\|\, m)))$
9       $\mathrm{T}_{\mathsf{h}}[Y, M] \leftarrow z$
10  else if $\mathrm{T}_{\mathsf{h}}[Y] \neq \bot$ then $z \leftarrow \mathrm{T}_{\mathsf{h}}[Y]$
11  else $z \leftarrow\!\!\$\; \{0,1\}^{2k}$; $\mathrm{T}_{\mathsf{h}}[Y] \leftarrow z$
12  if $(z \in \mathcal{G}_{\mathtt{all}}$.nodes and $(y, z, m) \notin \mathcal{G}_{\mathtt{all}}$.edges$)$
13     or $M \neq M_{\mathtt{all}}$
14      $\mathsf{bad} \leftarrow \mathsf{true}$
15  add $(y, z, m)$ to $\mathcal{G}$.edges
16  add $(y, z, m)$ to $\mathcal{G}_{\mathtt{all}}$.edges
17  return $z$
```

Figure 10: Games G_2, G_3, and G_4 in the proof of Theorem 6.1. Highlighted code is changed from the previous game, and boxed code is present only in G_3 (and subsequent games). Algorithms not shown are unchanged from the previous game.

At this point, the adversary can no longer directly query random oracle H, so we allow the simulator to lazily sample the function. Also in this game, the simulator queries H on the path from IV to y in $\mathcal{G}_{\mathtt{all}}$ for all queries, not just private queries. If the path in $\mathcal{G}$ is different from the path in $\mathcal{G}_{\mathsf{pub}}$, then the bad flag will be set and the adversary will lose anyway. Therefore the view in any winning game is unchanged, and

$$\Pr[\mathrm{G}_4] = \Pr[\mathrm{G}_5].$$

In our next game G_6, we replace the sampling of z from the preimages of a random point y with sampling a uniformly random $2k$-bit string. The sampling will never fail to be uniform, which means the adversary can distinguish the game if it were to fail in G_5; from condition (1) we have that the probability of failure was at most ϵ per query. Otherwise, we have from condition (3) on Out that the statistical distance of the distribution $(z \leftarrow\!\!\$\; \mathsf{Out}^{-1}(y) : \; : y \leftarrow\!\!\$\; S)$ from the uniform distribution on $\{0,1\}^{2k}$ is at most ϵ. By a hybrid argument over the $\mathrm{Q}^{\mathcal{A}}_{\mathrm{PUB}} + \ell \mathrm{Q}^{\mathcal{A}}_{\mathrm{PRIV}}$ queries to the simulator, the probability that $\mathcal{A}$ can distinguish G_5 from G_6 is bounded above by $2(\mathrm{Q}^{\mathcal{A}}_{\mathrm{PUB}} + \ell \mathrm{Q}^{\mathcal{A}}_{\mathrm{PRIV}})\epsilon$.

Now that we are caching z in table T_{H} when the check of line 6 holdss true, it has become redundant to cache it in table T_{h}, so we stop doing this caching. We must be careful since table T_{H} is indexed by labels of the form $\mathsf{unpad}(M_{\mathtt{all}} \,\|\, m)$ where T_{h} was indexed by tuples $(Y, M_{\mathtt{all}})$. Since $M_{\mathtt{all}}$ is a path from IV to y in a graph with no duplicate edges provided bad is not set, $M_{\mathtt{all}}$ uniquely determines its ending node y and $\mathsf{unpad}(M_{\mathtt{all}} \,\|\, m)$ uniquely determines a tuple $((y, m), M_{\mathtt{all}})$ because unpad is injective. Thus the entries of T_{H} are in one-to-one correlation with the entries of T_{h}, and we can safely retain only the former, and

$$\Pr[\mathrm{G}_6] \leq \Pr[\mathrm{G}_5] + 2(\mathrm{Q}^{\mathcal{A}}_{\mathrm{PUB}} + \ell \mathrm{Q}^{\mathcal{A}}_{\mathrm{PRIV}})\epsilon$$

```
Game G_5

S(Y, G):
1  $(y, m) \leftarrow Y$
2  if $\exists z$ such that $(y, z, m) \in \mathcal{G}$.edges
3     return $z$
4  $M \leftarrow \mathcal{G}$.FindPath$(IV, y)$
5  $M_{\mathtt{all}} \leftarrow \mathcal{G}_{\mathtt{all}}$.FindPath$(IV, y)$
6  if $M_{\mathtt{all}} \neq \bot$
         and $\mathsf{unpad}(M_{\mathtt{all}} \parallel m) \neq \bot$ then
7     if $\mathrm{T}_{\mathsf{h}}[Y, M_{\mathtt{all}}] \neq \bot$ then
8        $z \leftarrow \mathrm{T}_{\mathsf{h}}[Y, M_{\mathtt{all}}]$
9     else
10       if $\mathrm{T}_{\mathsf{H}}[\mathsf{unpad}(M_{\mathtt{all}} \parallel m)] \neq \bot$
11          $y \leftarrow \mathrm{T}_{\mathsf{H}}[\mathsf{unpad}(M_{\mathtt{all}} \parallel m)]$
12       $\mathrm{T}_{\mathsf{H}}[\mathsf{unpad}(M_{\mathtt{all}} \parallel m)] \leftarrow y$
13       $z \leftarrow\!\!\$\ \mathsf{Out}^{-1}(y)$; $\mathrm{T}_{\mathsf{h}}[Y, M_{\mathtt{all}}] \leftarrow z$
14 else if $\mathrm{T}_{\mathsf{h}}[Y] \neq \bot$ then $z \leftarrow \mathrm{T}_{\mathsf{h}}[Y]$
15 else $z \leftarrow\!\!\$\ \{0,1\}^{2k}$; $\mathrm{T}_{\mathsf{h}}[Y] \leftarrow z$
16 if $(z \in \mathcal{G}_{\mathtt{all}}$.nodes and $(y, z, m) \notin \mathcal{G}_{\mathtt{all}}$.edges$)$
17    or $M \neq M_{\mathtt{all}}$
18       $\mathsf{bad} \leftarrow \mathsf{true}$
19 add $(y, z, m)$ to $\mathcal{G}$.edges
20 add $(y, z, m)$ to $\mathcal{G}_{\mathtt{all}}$.edges
21 return $z$
```

```
Game G_6

S(Y, G):
1  $(y, m) \leftarrow Y$
2  if $\exists z$ such that $(y, z, m) \in \mathcal{G}$.edges
3     return $z$
4  $M \leftarrow \mathcal{G}$.FindPath$(IV, y)$
5  $M_{\mathtt{all}} \leftarrow \mathcal{G}_{\mathtt{all}}$.FindPath$(IV, y)$
6  if $M_{\mathtt{all}} \neq \bot$ and $\mathsf{unpad}(M_{\mathtt{all}} \parallel m) \neq \bot$ then
7     $z \leftarrow\!\!\$\ \{0,1\}^{2k}$
8     if $\mathrm{T}_{\mathsf{H}}[\mathsf{unpad}(M_{\mathtt{all}} \parallel m)] \neq \bot$
9        $z \leftarrow \mathrm{T}_{\mathsf{H}}[\mathsf{unpad}(M_{\mathtt{all}} \parallel m)]$
10    $\mathrm{T}_{\mathsf{H}}[\mathsf{unpad}(M_{\mathtt{all}} \parallel m)] \leftarrow z$
11 else if $\mathrm{T}_{\mathsf{h}}[Y] \neq \bot$ then $z \leftarrow \mathrm{T}_{\mathsf{h}}[Y]$
12 else $z \leftarrow\!\!\$\ \{0,1\}^{2k}$; $\mathrm{T}_{\mathsf{h}}[Y] \leftarrow z$
13 if $(z \in \mathcal{G}_{\mathtt{all}}$.nodes and $(y, z, m) \notin \mathcal{G}_{\mathtt{all}}$.edges$)$
14    or $M \neq M_{\mathtt{all}}$
15       $\mathsf{bad} \leftarrow \mathsf{true}$
16 add $(y, z, m)$ to $\mathcal{G}$.edges
17 add $(y, z, m)$ to $\mathcal{G}_{\mathtt{all}}$.edges
18 return $z$
```

Figure 11: Left: Game G_5 in the proof of Theorem 6.1. Right: Game G_6 in the proof of Theorem 6.1. Highlighted code is changed from the previous game, and algorithms not shown are unchanged from the previous game.

In G_7, all queries are sampled randomly from $\{0,1\}^{2k}$ and cached in table T_{h} under the input Y, instead of some being cached under the message $\mathsf{unpad}(M_{\mathtt{all}} \parallel m)$. We claim that in G_6 if a query $\mathcal{S}(y, m)$ stores z in $\mathrm{T}_{\mathsf{H}}[X]$, then a later query $\mathcal{S}(y', m')$ will return z if and only if $(y, m) = (y', m')$ or bad is set. The forward direction is trivial. If $\mathcal{S}(y', m')$ returns $\mathrm{T}_{\mathsf{H}}[X]$, then either we have

$$X = \mathsf{unpad}(\mathcal{G}_{\mathtt{all}}.\mathrm{FindPath}(IV, y') \parallel m') = \mathsf{unpad}(\mathcal{G}_{\mathtt{all}}.\mathrm{FindPath}(IV, y) \parallel m),$$

or there was a bad-setting collision between $\mathrm{T}_{\mathsf{H}}[X]$ and the randomly-sampled response z.

In the former case, the function unpad is injective, so we know $m = m'$, and the paths from IV to y' and y' respectively have the same sequence of edge labels. Unless bad is set, there are no duplicate edges, so a starting node and sequence of edge labels uniquely identify the ending node on the path; consequently $y = y'$ and the claim follows.

Queries in G_7 therefore hit a cache indexed by Y if and only if they would hit a cache indexed by X in G_6. We do not need to worry that the new entries in T_{h} overlap with those created in line 11; if the check in line 6 holds true during some query, then it cannot have been false in an earlier query with the same Y unless bad would be set. Thus no queries are answered from table T_{h} in

```
Game G_7

S(Y, G):
 1  $(y, m) \leftarrow Y$
 2  if $\exists z$ such that $(y, z, m) \in \mathcal{G}$.edges
 3      return $z$
 4  $M \leftarrow \mathcal{G}$.FindPath$(IV, y)$
 5  $M_{\texttt{all}} \leftarrow \mathcal{G}_{\texttt{all}}$.FindPath$(IV, y)$
 6  if $M_{\texttt{all}} \neq \bot$ and $\mathsf{unpad}(M_{\texttt{all}} \,\|\, m) \neq \bot$ then
 7      $z \leftarrow\!\!\$\ \{0,1\}^{2k}$
 8      if $\mathrm{T}_{\mathsf{h}}[Y] \neq \bot$ then $z \leftarrow \mathrm{T}_{\mathsf{h}}[Y]$
 9      $\mathrm{T}_{\mathsf{h}}[Y] \leftarrow z$
10  else if $\mathrm{T}_{\mathsf{h}}[Y] \neq \bot$ then $z \leftarrow \mathrm{T}_{\mathsf{h}}[Y]$
11  else $z \leftarrow\!\!\$\ \{0,1\}^{2k}$; $\mathrm{T}_{\mathsf{h}}[Y] \leftarrow z$
12  if $z \in \mathcal{G}_{\texttt{all}}$.nodes or $M \neq M_{\texttt{all}}$
13      bad ← true
14  add $(y, z, m)$ to $\mathcal{G}$.edges
15  add $(y, z, m)$ to $\mathcal{G}_{\texttt{all}}$.edges
16  return $z$
```

```
Game G_8

S(Y):
 1  if $\mathrm{T}_{\mathsf{h}}[Y] \neq \bot$ then $z \leftarrow \mathrm{T}_{\mathsf{h}}[Y]$
 2  else $z \leftarrow\!\!\$\ \{0,1\}^{2k}$; $\mathrm{T}_{\mathsf{h}}[Y] \leftarrow z$
 3  return $z$

FIN(c'):
 1  return $c'$
```

Figure 12: Left: Game G_7 in the proof of Theorem 6.1. Right: Game G_8 in the proof of Theorem 6.1. Highlighted code is changed from the previous game, and algorithms not shown are unchanged from the previous game.

G_7 that would not have been cached in earlier games, and

$$\Pr[\mathrm{G}_7] = \Pr[\mathrm{G}_6].$$

Notice that both branches of the simulator now identically sample $z \leftarrow\!\!\$\ \{0,1\}^{2k}$ uniformly, subject to caching in table T_{h} under Y; in the next game we will eliminate the redundant check on M_{all} in line 6.

In our final game, G_8, we remove the bad flag and the internal variables used to set it. This increases the adversary's advantage, since it can now win even if the game would set bad. The probability of a collision among the $\mathrm{Q}^{\mathcal{A}}_{\mathrm{PUB}} + \ell \mathrm{Q}^{\mathcal{A}}_{\mathrm{PRIV}}$ randomly sampled nodes of $\mathcal{G}_{\texttt{all}}$ is at most $\frac{(\mathrm{Q}^{\mathcal{A}}_{\mathrm{PUB}} + \ell \mathrm{Q}^{\mathcal{A}}_{\mathrm{PRIV}})^2}{2^{2k}}$ by a birthday bound. The probability that $\mathcal{G}_{\texttt{all}}$ contains a path to y that $\mathcal{G}_{\mathsf{pub}}$ does not is the probability that the adversary $\mathcal{A}$ queries PUB on one of the ℓq_{PRIV} intermediate nodes on a path in $\mathcal{G}_{\texttt{all}}$, before it learns the label of that node from PUB. $\mathcal{A}$ may use PRIV to learn the output y of Out an intermediate node, but it does not learn anything about which of the equally likely preimages of y is the label; from condition (2) we have that there are at least γ such preimages to guess from. Then the probability that $\mathcal{A}$ sets bad with a single PUB query is at most $\frac{\ell \mathrm{Q}^{\mathcal{A}}_{\mathrm{PRIV}}}{\gamma}$; a union bound over all PUB queries gives that a path exists in $\mathcal{G}_{\texttt{all}}$ but not $\mathcal{G}_{\mathsf{pub}}$ with probability no greater than $\frac{\mathrm{Q}^{\mathcal{A}}_{\mathrm{PUB}} \cdot \ell \mathrm{Q}^{\mathcal{A}}_{\mathrm{PRIV}}}{\gamma}$.

We also stop maintaining the graphs $\mathcal{G}_{\mathsf{pub}}$ and $\mathcal{G}_{\texttt{all}}$, which are now only used to cache queries whose responses are already cached in table T_{h}. This changes nothing about the view of the adversary, so

$$\Pr[\mathrm{G}_8] \leq \Pr[\mathrm{G}_7] + \frac{(\mathrm{Q}^{\mathcal{A}}_{\textsc{Pub}} + \ell \mathrm{Q}^{\mathcal{A}}_{\textsc{Priv}})^2}{2^{2k}} + \frac{\mathrm{Q}^{\mathcal{A}}_{\textsc{Pub}} \cdot \ell \mathrm{Q}^{\mathcal{A}}_{\textsc{Priv}}}{\gamma}.$$

If we look closely at G_8, we can see that the "simulator" is actually just a lazily-sampled random function with domain $\{0,1\}^{6k}$ and codomain $\{0,1\}^{2k}$. In fact, G_8 is identical to the "real" indifferentiability game for functor $\mathbf{F}$, save for its choice of challenge bit. Thus

$$\Pr[\mathrm{G}_8] = \Pr[\mathbf{G}^{\mathrm{indiff}}_{\mathbf{F},\mathcal{S}}(\mathcal{A}) | c = 1].$$

Collecting bounds across all gamehops gives the theorem.▌

Acknowledgments

We thank the (anonymous) reviewers of Crypto 2022, Asiacrypt 2022 and CT-RSA 2023 for their valuable comments. We thank Joseph Jaeger for his helpful comments and discussions about the correctness of chop-MD proofs in the literature.

References

[1] M. Abdalla, J. H. An, M. Bellare, and C. Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 418–433. Springer, Heidelberg, Apr. / May 2002. 4, 5, 6, 7, 14, 15, 20

[2] M. Backendal, M. Bellare, J. Sorrell, and J. Sun. The fiat-shamir zoo: Relating the security of different signature variants. In N. Gruschka, editor, *Secure IT Systems - 23rd Nordic Conference, NordSec 2018*, volume 11252 of *Lecture Notes in Computer Science*, pages 154–170. Springer, 2018. 14

[3] M. Bellare, D. J. Bernstein, and S. Tessaro. Hash-function based PRFs: AMAC and its multi-user security. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 566–595. Springer, Heidelberg, May 2016. 6, 17, 21

[4] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15. Springer, Heidelberg, Aug. 1996. 3

[5] M. Bellare and W. Dai. The multi-base discrete logarithm problem: Tight reductions and non-rewinding proofs for Schnorr identification and signatures. In K. Bhargavan, E. Oswald, and M. Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 529–552. Springer, Heidelberg, Dec. 2020. 5, 7, 15, 20

[6] M. Bellare, H. Davis, and F. Günther. Separate your domains: NIST PQC KEMs, oracle cloning and read-only indifferentiability. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 3–32. Springer, Heidelberg, May 2020. 6, 8

[7] M. Bellare and A. Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 162–177. Springer, Heidelberg, Aug. 2002. 5

[8] M. Bellare, B. Poettering, and D. Stebila. From identification to signatures, tightly: A framework and generic transforms. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 435–464. Springer, Heidelberg, Dec. 2016. 4, 7

[9] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993. 3, 8, 9

[10] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006. 7, 12

[11] M. Bellare and B. Tackmann. Nonce-based cryptography: Retaining security when randomness fails. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 729–757. Springer, Heidelberg, May 2016. 4, 7

[12] D. J. Bernstein. Multi-user Schnorr security, revisited. Cryptology ePrint Archive, Report 2015/996, 2015. `https://eprint.iacr.org/2015/996`. 14

[13] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2(2):77–89, 2012. 3, 4, 5, 14

[14] D. Bleichenbacher. A forgery attack on RSA signatures based on implementation errors in the verification. Rump Session Presentation, Crypto 2006, August 2006. 3

[15] J. Brendel, C. Cremers, D. Jackson, and M. Zhao. The provable security of Ed25519: Theory and practice. In *2021 IEEE Symposium on Security and Privacy*, pages 1659–1676. IEEE Computer Society Press, May 2021. 3, 4, 5, 6, 7, 14, 15, 16, 20

[16] K. Chalkias, F. Garillot, and V. Nikolaenko. Taming the many eddsas. In T. van der Merwe, C. Mitchell, and M. Mehrnezhad, editors, *Security Standardisation Research*, pages 67–90, Cham, 2020. Springer International Publishing. 6

[17] Y. Chen, A. Lombardi, F. Ma, and W. Quach. Does fiat-shamir require a cryptographic hash function? In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 334–363, Virtual Event, Aug. 2021. Springer, Heidelberg. 6

[18] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, Heidelberg, Aug. 2005. 3, 4, 6, 20, 21, 22

[19] I. Damgård. A design principle for hash functions. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 416–427. Springer, Heidelberg, Aug. 1990. 3, 5, 8

[20] Y. Dodis, T. Ristenpart, and T. Shrimpton. Salvaging Merkle-Damgård for practical applications. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 371–388. Springer, Heidelberg, Apr. 2009. 3, 4, 5, 18

[21] G. Fuchsbauer, A. Plouviez, and Y. Seurin. Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 63–95. Springer, Heidelberg, May 2020. 5, 15, 20

[22] O. Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In A. M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 104–110. Springer, Heidelberg, Aug. 1987. 4, 7

[23] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, Apr. 1988. 4, 9

[24] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In T. Kohno, editor, *USENIX Security 2012*, pages 205–220. USENIX Association, Aug. 2012. 3

[25] IANIX. Things that use Ed25519. `https://ianix.com/pub/ed25519-deployment.html`. 3, 4

[26] S. Josefsson and I. Liusvaara. Edwards-curve digital signature algorithm (EdDSA). RFC 8032, Jan. 2017. `https://datatracker.ietf.org/doc/html/rfc8032`. 3, 4

[27] E. Kiltz, D. Masny, and J. Pan. Optimal security proofs for signatures from identification schemes. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 33–61. Springer, Heidelberg, Aug. 2016. 6

[28] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, whit is right. Cryptology ePrint Archive, Report 2012/064, 2012. `https://eprint.iacr.org/2012/064`. 3

[29] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In M. Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, Feb. 2004. 4, 6, 17, 18, 20

[30] R. C. Merkle. A certified digital signature. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, Heidelberg, Aug. 1990. 3, 5, 8

[31] A. Mittelbach and M. Fischlin. *The Theory of Hash Functions and Random Oracles*. Springer Nature, Switzerland, 2021. 6, 21, 22

[32] D. M'Raïhi, D. Naccache, D. Pointcheval, and S. Vaudenay. Computational alternatives to random number generators. In S. E. Tavares and H. Meijer, editors, *SAC 1998*, volume 1556 of *LNCS*, pages 72–80. Springer, Heidelberg, Aug. 1999. 4, 7

[33] National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS PUB 186-5, Oct. 2019. `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5-draft.pdf`. 3, 4

[34] G. Neven, N. P. Smart, and B. Warinschi. Hash function requirements for schnorr signatures. *Journal of Mathematical Cryptology*, 3(1):69–87, 2009. 6

[35] K. Ohta and T. Okamoto. On concrete security treatment of signatures derived from identification. In H. Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 354–369. Springer, Heidelberg, Aug. 1998. 6

[36] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. 4, 5, 6, 7, 15, 20

[37] T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, Heidelberg, May 2011. 4

[38] L. Rotem and G. Segev. Tighter security for schnorr identification and signatures: A high-moment forking lemma for *Σ*-protocols. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 222–250, Virtual Event, Aug. 2021. Springer, Heidelberg. 5, 15, 20

[39] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, Jan. 1991. 4, 5, 14

[40] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. 6

[41] K. Yoneyama, S. Miyagawa, and K. Ohta. Leaky random oracle. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 92(8):1795–1807, 2009. 4, 5

A The unique order-p subgroup of $\mathbb{G}$

Here, we briefly prove that our choice in Section 2 of $\mathbb{G}_{\mathsf{p}}$ as the unique subgroup of order p of group $\mathbb{G}$, which has order $\mathsf{p} \cdot 2^{\mathsf{f}}$, is well-defined. (We do not prove that $\mathbb{G}_{\mathsf{p}}$ is cyclic as this follows directly from the fact that its order is prime.) We also give an efficient test for membership in $\mathbb{G}_{\mathsf{p}}$.

Proposition A.1 Let p be an odd prime, let $2^{\mathsf{f}} < \mathsf{p}$ be a positive integer, and let $\mathbb{G}$ be a group of order $2^{\mathsf{f}} \cdot \mathsf{p}$. Then (1) the group $\mathbb{G}$ has a unique subgroup of order p, and (2) For all $X \in \mathbb{G}$ it is the case that X is in this subgroup iff $\mathsf{p} \cdot X = 0_{\mathbb{G}}$.

(1) Let n be the number of p-order subgroups of $\mathbb{G}$. According to Sylow's theorem $n \equiv 1 \mod \mathsf{p}$. We now have two cases: either $n = 1$, or $n > 1$. We prove that $n = 1$ by contradiction; therefore we assume $n > 1$. It follows that $n \geq \mathsf{p}+1$. Two distinct groups of prime order can intersect only at the identity, so each of the n subgroups of $\mathbb{G}$ contains $\mathsf{p}-1$ unique elements. Consequently the order of $\mathbb{G}$ is at least $n(\mathsf{p}-1) \geq (\mathsf{p}+1)(\mathsf{p}-1) \geq \mathsf{p}(\mathsf{p}+1)$. Since we have already defined the order of $\mathbb{G}$ to be $2^{\mathsf{f}} \cdot \mathsf{p}$, we have that $2^{\mathsf{f}} \geq \mathsf{p}+1$. This contradicts our initial assumption that $2^{\mathsf{f}} < \mathsf{p}$; thus our assumption that $n > 1$ must be false and $\mathbb{G}$ must have exactly one subgroup of order p. This subgroup is $\mathbb{G}_{\mathsf{p}}$.

(2) Let $X \in \mathbb{G}$ be a group element and assume that $\mathsf{p} \cdot X = 0_{\mathbb{G}}$. This implies that the order of X divides p. Since p is prime, either the order of X is 1 or it is p. In the first case, $x = 0_{\mathbb{G}}$. Otherwise, X generates a subgroup with order p, which by part (1) is the unique such subgroup $\mathbb{G}_{\mathsf{p}}$. Therefore X generates $\mathbb{G}_{\mathsf{p}}$ and must belong to it.

For the reverse direction, assume that X is in $\mathbb{G}^{\mathsf{p}}$. The order of X must divide the order of $\mathbb{G}_{\mathsf{p}}$; so X must either have order p or order 1. In either case, $\mathsf{p} \cdot X = 0_{\mathbb{G}}$.

B Group Instantiation for Ed25519

Ed25519 is EdDSA with twisted Edwards curve which is birationally equivalent to the curve Curve25519. Curve25519 is of Montgomery form with equation $v^2 = u^3 + 486662u^2 + u$ over the field $\mathbb{F}_q$, and it is birationally equivalent to the Edwards curve $x^2 + y^2 = 1 + (121665/121666)x^2y^2$ where $x = \sqrt{486664}u/v$ and $y = (u-1)/(u+1)$.

In general EdDSA, the group is the set of points on an Edwards curve E, namely the $E(\mathbb{F}_q) = \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q : -x^2 + y^2 = 1 + dx^2y^2\}$. The Edwards curve in Ed25519 is isomorphic to $-x^2 + y^2 = 1 - (121665/121666)x^2y^2$, and so it has $d = -(121665/121666) \in \mathbb{F}_q$. Then defines $\mathbb{G}$ to be $E(\mathbb{F}_q) = \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q : -x^2 + y^2 = 1 - (121665/121666)x^2y^2\}$. The field size q is the prime $2^{255} - 19$. Other parameters of the group descriptor include the following: $\mathsf{f} = 3$ is the $\log_2$ of cofactor 2^{f}; $\mathsf{p} = 2^{252} + 27742317777372353535851937790883648493$ is the odd prime order of subgroup $\mathbb{G}_{\mathsf{p}}$; $\mathsf{B} = (x, 4/5) \in E$ is the generator of $\mathbb{G}_{\mathsf{p}}$, and $\mathsf{p} \cdot \mathsf{B} = 0$. The exact implementation of Ed25519 also fixes $b = 256$ and FO function to be SHA-512 such that the output of FO is $2b$ bits.

For an elliptic curve point (x, y), the encoding function en encodes it as the $(b-1)$-bit little-endian encoding of y followed by a sign bit of x. The sign bit is 1 if and only if x is negative and x is negative if its $(b-1)$-bit encoding is lexicographically larger than that of $-x$. The output length el is therefore $b = 256$. For decoding, de recovers y immediately from its $(b-1)$-bit encoding, and then recovers x via $x = \pm\sqrt{(y^2-1)/(dy^2+1)}$ and the sign bit. If the resulting point is not on the curve or if taking the square root fails, de returns $\perp$.