On the security of ECDSA with additive key derivation and presignatures

Jens Groth and Victor Shoup
DFINITY

April 27, 2022

Abstract

Two common variations of ECDSA signatures are *additive key derivation* and *presignatures*. Additive key derivation is a simple mechanism for deriving many subkeys from a single master key, and is already widely used in cryptocurrency applications with the Hierarchical Deterministic Wallet mechanism standardized in Bitcoin Improvement Proposal 32 (BIP32). Because of its linear nature, additive key derivation is also amenable to efficient implementation in the threshold setting. With presignatures, the secret and public nonces used in the ECDSA signing algorithm are precomputed. In the threshold setting, using presignatures along with other precomputed data allows for an extremely efficient "online phase" of the protocol. Recent works have advocated for both of these variations, sometimes combined together. However, somewhat surprisingly, we are aware of no prior security proof for additive key derivation, let alone for additive key derivation in combination with presignatures.

In this paper, we provide a thorough analysis of these variations, both in isolation and in combination. Our analysis is in the generic group model (GGM). Importantly, we do not modify ECDSA or weaken the standard notion of security in any way. Of independent interest, we also present a version of the GGM that is specific to elliptic curves. This EC-GGM better models some of the idiosyncrasies (such as the conversion function and malleability) of ECDSA. In addition to this analysis, we report security weaknesses in these variations that apparently have not been previously reported. For example, we show that when both variations are combined, there is a cube-root attack on ECDSA, which is much faster than the best known, square-root attack on plain ECDSA. We also present two mitigations against these weaknesses: re-randomized presignatures and homogeneous key derivation. Each of these mitigations is very lightweight, and when used in combination, the security is essentially the same as that of plain ECDSA (in the EC-GGM).

1 Introduction

Let us recall the basic ECDSA signature scheme [NIST13]. Let E be an elliptic curve defined over $\mathbb{Z}_p$ and generated by a point $\mathcal{G}$ of prime order q, and let E^* be the set of points (x, y) on the curve excluding the point at infinity $\mathcal{O}$. The **unreduced conversion function** $C : E^* \to \mathbb{Z}_p$ maps a point $\mathcal{P}$ to its x-coordinate. The **reduced conversion function** $\bar{C} : E^* \to \mathbb{Z}_q$ maps a point $\mathcal{P}$ to the canonical representative of $C(\mathcal{P})$ (i.e., an integer in the range $[0, p)$) reduced mod q.

```
Sign message $m$:
    $h \leftarrow \mathit{Hash}(m) \in \mathbb{Z}_q$
    $r \xleftarrow{\$} \mathbb{Z}_q^*,\ \mathcal{R} \leftarrow r\mathcal{G} \in E,\ t \leftarrow \bar{C}(\mathcal{R}) \in \mathbb{Z}_q$
    if $t = 0$ or $h + td = 0$ then return *fail*
    $s \leftarrow r^{-1}(h + td)$
    return the signature $(s, t)$
```

```
Verify signature $(s, t) \in \mathbb{Z}_q^* \times \mathbb{Z}_q^*$ on $m$:
    $h \leftarrow \mathit{Hash}(m) \in \mathbb{Z}_q$
    $\mathcal{R} \leftarrow s^{-1}h\mathcal{G} + s^{-1}t\mathcal{D}$
    check that $\mathcal{R} \neq \mathcal{O}$ and $\bar{C}(\mathcal{R}) = t$
```

Figure 1: ECDSA signing and verification algorithms

The secret key for ECDSA is a random $d \in \mathbb{Z}_q^*$, the public key is $\mathcal{D} = d\mathcal{G} \in E$. The scheme makes use of a hash function $\mathit{Hash} : \{0,1\}^* \to \mathbb{Z}_q$. The signing and verification algorithms are shown in Figure 1.

Note that the signing algorithm will *fail* with only negligible probability (as discussed below in Section 3). Also note for a valid signature (s, t) on a message m, we have

$$h + td = sr,$$

where $h := \mathit{Hash}(m)$, or equivalently,

$$h\mathcal{G} + t\mathcal{D} = s\mathcal{R}.$$

The security of ECDSA has only been analyzed in idealized models of computation. Specifically, Brown [Bro02] showed that under standard intractability assumptions on *Hash* (collision resistance and random/zero preimage resistance), ECDSA is secure in the generic group model [Nec94, Sho97]. In addition, Fersch, Kiltz, and Pottering [FKP16] have also showed that ECDSA is secure under somewhat different intractability assumptions on *Hash* if the conversion function is modeled as an idealized function (but one that captures some idiosyncrasies of the actual conversion function). In this paper, we will also analyze ECDSA and several variants in the generic group model. However, we shall work in a specific version of the generic group model that more accurately models some of the idiosyncrasies of elliptic curves and the corresponding conversion function. We call this the **elliptic curve generic group model (EC-GGM)**, which may be of independent interest. By working in this model, we overcome objections raised in [FKP16] and elsewhere [SPMS02] that Brown's analysis was incomplete. For example, it was pointed out that Brown's analysis ruled out any malleability in the signature scheme, whereas ECDSA signatures are in fact malleable.

Several variations of ECDSA have been proposed, notably **additive key derivation** and **presignatures**. We are mainly interested in these variations because of the optimizations they enable in the threshold setting, where the signing functionality is implemented as a secure distributed protocol by parties that each hold a share of the secret key. However, these variations also enable optimizations in the single-signer setting as well.

Additive key derivation. With additive key derivation, the secret-key/public-key pair $(d, \mathcal{D})$ is viewed as a **master key pair** from which **subkey pairs** can be derived using a simple additive shift. Specifically, we can derive a secret subkey of the form $d + e$ by using a "tweak" $e \in \mathbb{Z}_q$. For such a derived secret subkey, we can compute the corresponding derived public subkey from the public key $\mathcal{D}$ as $\mathcal{D}+e\mathcal{G}$. In the context of cryptocurrency, this type of additive key derivation is used in so-called **Hierarchical Deterministic Wallets** using the Bitcoin Improvement Proposal 32 (BIP32) standard [Wui20], which is a specific way of deriving a tweak e via a chain of hashes applied to the public key and other public data. Note that BIP32 also specifies so-called "hardened" subkeys, which derives subkeys using the secret key — we do not consider such "hardened" subkeys in this paper.

There is a cost to storing secret keys, and additive key derivation is useful in reducing that cost, since it allows several distinct public keys to be used while only having to store a single secret key. This secret-key storage cost manifests itself in both the threshold and non-threshold settings. In the non-threshold setting, there is the obvious cost of maintaining the secret key in some kind of secure storage. In the threshold setting, there is the cost of running the key generation algorithm and storing secret shares in some kind of secure storage. There may be additional costs in the threshold setting: for example, the cost of resharing the secret key periodically, both to provide proactive security and to allow for dynamic changes in the share-holder membership. Because of the linearity of the key derivation, implementing additive key derivation in the threshold setting comes at essentially no cost.

Unfortunately, and somewhat surprisingly, we are aware of no prior proofs of security for ECDSA with additive key derivation. While [YY19] purports to present such a proof (via a direct reduction to the security of ECDSA), their proof seems to be fundamentally flawed: their simulator apparently needs to "reprogram" a random oracle that has already been "programmed". The more recent work [DEF$^+$21] analyzes additive key derivation with respect to a variant of ECDSA in which the derived public key is prepended to the message to be signed, and with a restricted attack model in which an attacker is only allowed to ask for one signature per message and derived public key.

Presignatures. In the signing algorithm, the values r and $\mathcal{R} := r\mathcal{G}$ are independent of the message to be signed (or the tweak), and so they can be precomputed in advance of an actual signing request. In the threshold setting, it is tempting to not only precompute a sharing of r, but to also to precompute $\mathcal{R}$ itself. This can greatly simplify the online signing phase of the protocol. Indeed, several papers, including [DJN$^+$20] and [GG20] present protocols that use presignatures. Moreover, [DJN$^+$20] advocates for the combination of presignatures and additive key derivation, even though the security of additive key derivation, let alone additive key derivation in combination with presignatures, has never been analyzed.

The paper [CMP20] does consider the security of presignatures (in isolation). They give an explicit definition and they briefly sketch a proof of security in the GGM with *Hash* also modeled as a random oracle. (an earlier version of [CMP20] had an incorrect security bound).

1.1 Our contributions

1.1.1 Security proofs

We carry out a careful and detailed security analysis of ECDSA and several variants, including ECDSA with additive key derivation, ECDSA with presignatures, and ECDSA with both additive key derivation and presignatures. This analysis is done in the generic group model (more precisely, the EC-GGM) under concrete assumptions for the hash function *Hash*. Importantly, we do not modify ECDSA or weaken the standard notion of security in any way. Unlike [CMP20], we do not model *Hash* as a random oracle, and we give precise security bounds. Our analysis carries over immediately to any threshold implementation of ECDSA whose security reduces to that of the non-threshold scheme (which is typically the case).

For additive key derivation, we mainly assume that the set $\mathfrak{E}$ of all valid tweaks is not too large and is determined in advance. In practice (such as with BIP32), tweaks are derived, via a hash, from identifiers (possibly combined with a "root" public key). This assumption on $\mathfrak{E}$ can be justified if the set of valid identifiers, and in particular, the set of identifiers with respect to which we are concerned about forgeries, is indeed small. It can also be further justified by modeling the hash function used to derive tweaks as a random oracle. That said, our analysis also works without this assumption, and we describe how our security results can be stated in terms of concrete security properties of the hash used to derive the tweaks. In Appendix D we provide an analysis of the BIP32 key derivation function, which justifies modeling it as a (public use) random oracle.

1.1.2 Attacks

While we are able to prove security results under reasonable assumptions for all of the variations listed above, in the course of our analysis, we discovered that the concrete security of some of these variants is substantially worse than plain ECDSA.

An attack on ECDSA with additive key derivation and presignatures. For example, consider ECDSA with both additive key derivation and presignatures. Consider the following attack:

1. Make one presignature query to get the group element $\mathcal{R}$ and let $t := \bar{C}(\mathcal{R})$.
2. Find m, e, m^*, e^* such that $h + te = h^* + te^*$, where $e \neq e^*$ and $h := \mathit{Hash}(m)$ and $h^* := \mathit{Hash}(m^*)$
3. Ask for a signature (s, t) using this presignature on message m with tweak e.

Then we have

$$\begin{aligned}\mathcal{R} &= s^{-1}h\mathcal{G} + s^{-1}t(\mathcal{D} + e\mathcal{G})\\ &= s^{-1}(h + te)\mathcal{G} + s^{-1}t\mathcal{D}\\ &= s^{-1}(h^* + te^*)\mathcal{G} + s^{-1}t\mathcal{D}\\ &= s^{-1}h^*\mathcal{G} + s^{-1}t(\mathcal{D} + e^*\mathcal{G}),\end{aligned}$$

which means that (s, t) is also a valid signature on m^* with respect to e^*.

Also observe that Step 2 above is essentially a 4-sum problem of the type studied by Wagner [Wag02] and others [BLN+09, NS15]. Indeed, Wagner's algorithm allows us to implement Step 2 in time significantly less than $O(q^{1/2})$ if the set $\mathfrak{E}$ is sufficiently large. In particular, if $|\mathfrak{E}| = \Theta(q^{1/3})$, then we can solve this 4-sum problem and forge a signature in time roughly $O(q^{1/3})$. While not a polynomial-time attack, this is clearly a much more efficient attack than the best-known attack on plain ECDSA, which runs in time roughly $O(q^{1/2})$.

An attack on ECDSA with presignatures. Even with presignatures alone, ECDSA has potential security weaknesses that plain ECDSA does not. Consider the following attack:

1. Make one presignature query to get the group element $\mathcal{R}$ and let $t := \bar{C}(\mathcal{R})$.
2. Compute $\mathcal{R}^* \leftarrow c\mathcal{R}$ for some $c \in \mathbb{Z}_q^*$ and let $t^* := \bar{C}(\mathcal{R}^*)$.
3. Find m, m^* such that $h/t = h^*/t^*$, where $h := \mathit{Hash}(m)$ and $h^* := \mathit{Hash}(m^*)$ and $m \neq m^*$.
4. Ask for a signature (s, t) using the presignature with group element $\mathcal{R}$ on message m.
5. Compute s^* satisfying $(s^*)^{-1}t^* = cs^{-1}t$, and output (s^*, t^*).

We have

$$\mathcal{R} = s^{-1}h\mathcal{G} + s^{-1}t\mathcal{D}.$$

Moreover,

$$\begin{aligned}\mathcal{R}^* &= cR = cs^{-1}h\mathcal{G} + cs^{-1}t\mathcal{D} \\ &= cs^{-1}t(h/t)\mathcal{G} + cs^{-1}t\mathcal{D} \\ &= (s^*)^{-1}t^*(h/t)\mathcal{G} + (s^*)^{-1}t^*\mathcal{D} \\ &= (s^*)^{-1}t^*(h^*/t^*)\mathcal{G} + (s^*)^{-1}t^*\mathcal{D} \\ &= (s^*)^{-1}h^*\mathcal{G} + (s^*)^{-1}t^*\mathcal{D},\end{aligned}$$

which means that (s^*, t^*) is a valid signature on m^*.

To implement Step 3, for fixed t and t^*, there is no obvious way to find h, h^* satisfying $h/t = h^*/t^*$ in time faster than $O(q^{1/2})$. However, the inability to do so requires an assumption on *Hash* that is not needed for plain ECDSA. Moreover, it is clear that ECDSA with presignatures is *completely insecure* if we allow a “raw” signing oracle, i.e., a signing oracle that takes as input the purported hash h rather than the message m. There are settings where allowing such “raw” signing queries may be useful (e,g., in a remote signing service to avoid the cost of message transmission), and plain ECDSA is secure in the EC-GGM even with raw signing queries.

Note that one could extend the above attack so that the attack iterates Steps 3 and 4 for many values of c. This would give us an attack that is essentially a multiplicative variant of a 3-sum problem, for which there is no known algorithm that runs in time $O(q^{1-\epsilon})$ for any $\epsilon > 0$ [NS15]. However, this is again an attack vector that is not available for plain ECDSA.

1.1.3 Mitigations

In addition to the analysis and attacks discussed above, we introduce several mitigations.

Re-randomized presignatures. A presignature of the form $r' \in \mathbb{Z}_q$ and $\mathcal{R}' := r'\mathcal{G} \in E$ is computed as before. However, when a signing request is made, the actual presignature used is $r := r' + \delta$ and $\mathcal{R} := \mathcal{R}' + \delta\mathcal{G}$, where $\delta \in \mathbb{Z}_q$ is a public value that is pseudo-randomly generated at the time of the signing request (the key property is that δ is not predictable). This mitigation may be deployed both with and without additive key derivation.

We prove much stronger security results with this mitigation. Specifically, we prove a security result for re-randomized presignatures without additive key derivation that is essentially equivalent to the security result for plain ECDSA. With additive key derivation, the concrete security degrades by a factor of $|\mathfrak{E}|$, where $\mathfrak{E}$ is the set of valid tweaks, but the resulting scheme is no longer vulnerable to the 4-sum attack described above. Both with and without additive key derivation, we can also prove security even with respect to a raw signing oracle.

We are mainly interested in the use of re-randomized presignatures in the threshold setting. Since the re-randomization is linear, in terms of working with linear secret sharing, the impact is negligible (computing $(r' + \delta)^{-1}$ in the threshold setting is no harder than computing r^{-1}, assuming one is using standard techniques, such as [BB89]). However, the parties will still need access to a source of public randomness to generate δ. Accessing this public randomness may or may not introduce some extra latency, depending on details of the system. For example, in the Internet Computer (IC) [DFI22] that motivated our work there is already a mechanism for accessing public, unpredictable randomness via a "random tape" (which is implemented using a threshold BLS signature [BLS01]). Moreover, in the IC architecture, when a subprotocol (such as a threshold ECDSA signing protocol) is launched, we can access this public randomness with no additional latency.

Instead of generating δ at the time of the signing request, as an alternative approach, one might also derive δ from a hash applied to (among other things) the public key, the (hash of) the message to be signed, and (if using additive key derivation) the tweak. This approach for re-randomizing presignatures comes at essentially no cost, either in terms of computation or latency. However, while it heuristically appears to offer more security than plain presignatures, and in particular foils the 4-sum attack described above, we have not formally analyzed the security of this approach.

Homogeneous key derivation. We also propose an alternative additive key derivation mechanism with better security properties. The master secret key now consists of a randomly chosen pair $(d, d') \in \mathbb{Z}_q \times \mathbb{Z}_q$. The corresponding master public key is

$$(\mathcal{D}, \mathcal{D}') := (d\mathcal{G}, d'\mathcal{G}).$$

Given a tweak $e \in \mathbb{Z}_q$, the derived secret key is $d+ed'$, and the derived public key is $\mathcal{D}+e\mathcal{D}'$.

Clearly, just as for additive key derivation, we can easily derive a public key from the master public key. Moreover, since key derivation is linear, implementing homogeneous key derivation in the threshold setting comes at very little cost. Compared to additive key

	no presigs	presigs	re-randomized presigs
no derivation	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ 🚫	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
additive	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + N_{\text{psig}}\,\mathcal{E}_{\text{4sum1}} + N\mathcal{E}_{\text{4sum2}} + \mathcal{E}_{\text{zpr}} + N^2/q$ 🚫	$\mathcal{E}_{\text{cr}} + N\lvert\mathfrak{E}\rvert\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$
homogeneous	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$	$\mathcal{E}_{\text{cr}} + UN\mathcal{E}_{\text{rpr}} + N\mathcal{E}_{\text{rr}} + \mathcal{E}_{\text{zpr}} + N^2/q$ 🚫	$\mathcal{E}_{\text{cr}} + N\mathcal{E}_{\text{rpr}} + \mathcal{E}_{\text{zpr}} + N^2/q$

Table 1: Summary of concrete security theorems

derivation, the only downsides are (1) some small additional computational and communication complexities, and (2) the lack of compatibility with existing standards, such as BIP32.

One can combine homogeneous key derivation with either plain ECDSA, ECDSA with presignatures, and ECDSA with re-randomized presignatures. We give security proofs for all three of these variations. The upshot is that with homogeneous key derivation, for each variation, we get a security result for that variation *with* homogeneous key derivation that is essentially equivalent to that variation *without* key derivation. In particular, unlike with additive key derivation, our security results do not degrade linearly with $\lvert\mathfrak{E}\rvert$, where $\mathfrak{E}$ is the set of valid tweaks, and we do not need to insist that the set $\mathfrak{E}$ is determined in advance. In particular, we may just assume that the tweaks are derived by a collision resistant hash.

1.1.4 Summary of concrete security bounds

Table 1 summarizes our concrete security theorems. Each table entry gives an upper bound on an adversary's success in producing a forgery (ignoring small constants) in the EC-GCM (and in the PDF file, each table entry also contains a hyperlink to the actual theorem). These upper bounds are stated in terms of:

- q: the order of the group E;
- N: the number of oracle queries (group, signing, or presignature);
- N_{psig}: the number of presignature requests;
- U: the maximum number of *unused* presignature requests outstanding at any point in time;
- $\lvert\mathfrak{E}\rvert$: the size of the set of valid tweaks;
- $\mathcal{E}_{\text{cr}}$: the probability of successfully finding a collision in *Hash*;
- $\mathcal{E}_{\text{rpr}}$: the probability of successfully finding a preimage under *Hash* of a random element in $\mathbb{Z}_q$;
- $\mathcal{E}_{\text{zpr}}$: the probability of successfully finding a preimage under *Hash* of 0;

- $\mathcal{E}_{\mathrm{rr}}$: the probability, given random $\rho \in \mathbb{Z}_q^*$, of finding m, m^* such that $h/h^* = \rho$, where $h := \mathit{Hash}(m)$ and $h^* := \mathit{Hash}(m^*) \neq 0$;
- $\mathcal{E}_{\mathrm{4sum1}}$: the probability, given random $t \in \mathbb{Z}_q$, of successfully finding m, e, m^*, e^* such that $h + te = h^* + te^*$, where $e, e^* \in \mathfrak{E}$, $e \neq e^*$ and $h := \mathit{Hash}(m)$ and $h^* := \mathit{Hash}(m^*)$;
- $\mathcal{E}_{\mathrm{4sum2}}$: the probability of successfully finding m, e, m^*, e^* such that $h/t + e = h^*/t^* + e^*$, where $e, e^* \in \mathfrak{E}$, $h := \mathit{Hash}(m)$, $h^* := \mathit{Hash}(m^*) \neq 0$, where $t \in \mathbb{Z}_q^*$ is selected by the adversary from one of several random samples, and $t^* \in \mathbb{Z}_q^*$ is a random value given after t is selected.

The success probabilities $\mathcal{E}_{\mathrm{cr}}, \mathcal{E}_{\mathrm{rpr}}, \mathcal{E}_{\mathrm{zpr}}, \mathcal{E}_{\mathrm{rr}}, \mathcal{E}_{\mathrm{4sum1}}, \mathcal{E}_{\mathrm{4sum2}}$ are stated in terms of an adversary whose running time is essentially that of the forging adversary (or that time plus UN, in either of the presignature settings). Also, the symbol 🚫 in the table indicates that this mode of operation is insecure with "raw" signing, i.e., with pre-signatures the adversary can forge if she can make signature queries directly on h instead of m, while the other variations permit raw signatures.

We make some quick observations about this table. First, observe that the first and third rows are identical, as are the first and third columns. Second, we see that the best security bounds are in the upper left cell and the lower right cell, and these bounds are the same — this suggests that ECDSA with homogeneous key derivation and re-randomized presignatures is just as secure as plain ECDSA. Third, we see that the worst security result is in the middle cell, corresponding to the setting of additive key derivation combined with (non-re-randomized) presignatures; moreover, this is not just a case of sloppy analysis, as we have already seen that in this setting, there is an actual attack that produces a forgery in time significantly faster than $O(q^{1/2})$. Finally, we see that "raw" signing is insecure for all modes of operation in the middle column. Each other mode are secure even with "raw" signing, meaning that the mode is just as secure if the signing algorithm is given an arbitrary hash value $h \in \mathbb{Z}_q$ (not necessarily the output of *Hash*) and, in the case of key derivation, and arbitrary tweak $e \in \mathbb{Z}_q$ (not necessarily in $\mathfrak{E}$ or satisfying any other constraint).

As we shall observe, all of our security proofs for the unforgeability of ECDSA and all of its variants also prove a stronger notion of security, which we call *strong unforgeability up to sign*. See Section 4.1.1 for more details.

2 The EC-GGM

We propose the following **elliptic curve generic group model (EC-GGM)**.

We assume an elliptic curve E is defined by an equation $y^2 = F(x)$ over $\mathbb{Z}_p$ and that the curve contains q points including the point at infinity $\mathcal{O}$. Here, p and q are odd primes. Let E^* be the set of non-zero points (excluding the point at infinity) on the curve, i.e., $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ that satisfy $y^2 = F(x)$. From now on, we shall not be making any use of the usual group law for E, but simply treat E as a set; however, for a point $\mathcal{P} = (x, y) \in E^*$, we write $-P$ to denote the point $(x, -y) \in E^*$. Note that because we are assuming q is prime, there are no points of the form $(x, 0) \in E$ (these would be points of order 2 under the usual group law).

An **encoding function** for E is a function

$$\pi : \mathbb{Z}_q \mapsto E$$

that is

- injective,
- **identity preserving**, meaning that $\pi(0) = \mathcal{O}$, and
- **inverse preserving**, meaning that for all $i \in \mathbb{Z}_q$, $\pi(-i) = -\pi(i)$.

In the EC-GGM, parties know E and interact with a **group oracle** $\mathcal{O}_{\text{grp}}$ that works as follows:

- $\mathcal{O}_{\text{grp}}$ on initialization chooses an encoding function π at random from the set of all encoding functions
- $\mathcal{O}_{\text{grp}}$ responds to two types of queries:
  - $(\texttt{map}, i)$, where $i \in \mathbb{Z}_q$:
    - return $\pi(i)$
  - $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2)$, where $\mathcal{P}_1, \mathcal{P}_2 \in E$:
    - return $\pi\big(\ \pi^{-1}(\mathcal{P}_1) + \pi^{-1}(\mathcal{P}_2)\ \big)$

NOTES:

1. The intuition is that the random choice of encoding function hides relations between group elements.

2. However, to make things more realistic in terms of the ECDSA conversion function, the encodings themselves have the same format as in a concrete elliptic curve, even though we do not at all use the group law of an elliptic curve.

3. Also to make things more realistic, the trivial relationship between a point and its inverse (that they share the same x-coordinate) is preserved.

4. Our model only captures the situation of elliptic curves over $\mathbb{Z}_p$ of prime order and cofactor 1. This is sufficient for many settings, and it covers all of the “`secp`” curves in [Cer10].

5. It would be possible to extend the model to elliptic curves of non-prime order as well, in which case the domain of the encoding function π would have to be adjusted to match the structure of the group.

3 Properties of the ECDSA conversion function

For a random variable T taking values in some finite set $\mathfrak{X}$, we define its **guessing probability** to be

$$\max\{\Pr[T = x] : x \in \mathfrak{X}\}.$$

Recall again the ECDSA signature scheme as described in Section 1 and Figure 1. The unreduced conversion function $C : E^* \to \mathbb{Z}_p$ is a 2-to-1 map (recall that there are no points of the form $(x, 0) \in E$). Therefore, the distribution of $C(\mathcal{R})$, for random $\mathcal{R} \in E^*$, is uniform over a subset of $\mathbb{Z}_p$ of size $(q-1)/2$. In particular, the guessing probability of $C(\mathcal{R})$ is $2/(q-1)$.

Hasse's theorem says that $q - 1 = p + 2\theta p^{1/2}$ for some $\theta \in [-1, 1]$. This implies that for $p \geq 13$, we have $p/2 \leq q \leq 2p$. We shall implicitly assume this from now on. The bound $p \leq 2q$ and the fact that C is 2-to-1 imply that every element of $\mathbb{Z}_q$ has at most four preimages under the reduced conversion function $\bar{C} : E^* \to \mathbb{Z}_q$; therefore, the guessing probability of $t := \bar{C}(\mathcal{R})$ is at most $4/(q-1)$. The ECDSA signing algorithm fails if $t = 0$ or $h + td = 0$. Thus, the probability that the signing algorithm fails is at most $8/(q-1)$.

Hasse's theorem also implies that the probability that $x \in C(E^*)$, for random $x \in \mathbb{Z}_p$, is equal to $1/2 + \theta p^{-1/2}$. We can use this to design an efficient probabilistic **sampling algorithm *Samp***, which takes as input $t \in \mathbb{Z}_q$ and returns either `fail` or a point $\mathcal{R} \in \bar{C}^{-1}(t)$, with the following properties:

- For randomly chosen $t \in \mathbb{Z}_q$, we have

$$\Pr[\mathit{Samp}(t) = \texttt{fail}] \leq \tfrac{3}{4} + \tfrac{1}{2}p^{-1/2}.$$

- For randomly chosen $t \in \mathbb{Z}_q$, the conditional distribution of *Samp*(t), given that *Samp*$(t) \neq$ `fail`, is uniform over E^*.

The algorithm works as follows:

1. Let $t' \in \mathbb{Z}$ be the canonical representative of t in the interval $[0, q)$. (*Assume t is uniform over $\mathbb{Z}_q$. t' is uniform over* $\{0, \ldots, q-1\}$.)
2. If $q < p$, then with probability $1/2$ add q to t'. (t' *is uniform over an interval* $\{0, \ldots, u-1\}$*, where* $p \leq u \leq 2p$.)
3. If $t' \geq p$ then return `fail`. (*Failure occurs with probability at most* $1/2$*; otherwise,* t' *is uniform over* $\{0, \ldots, p-1\}$.)
4. Set $x \leftarrow [t' \bmod p] \in \mathbb{Z}_p$. ($x$ *is uniform over* $\mathbb{Z}_p$.)
5. If $F(x)$ is not a square, return `fail`. (*Failure occurs with probability* $1/2 - \theta p^{-1/2}$.)
6. Choose a random square root y of $F(x)$ and return $\mathcal{R} := (x, y)$. ($\mathcal{R}$ *is uniform over* E^*.)

4 Notions of security

4.1 Signature schemes

Definition 1 (CMA security). *For a signature scheme $\mathcal{S}$ and an adversary $\mathcal{A}$, we denote by* $\mathrm{CMA}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ *the advantage that $\mathcal{A}$ has in forging a signature in a chosen message attack against $\mathcal{S}$. This is the probability that $\mathcal{A}$ wins the following game.*

- *The challenger runs the key generation algorithm for $\mathcal{S}$ to obtain a public key pk and a secret key sk and gives pk to $\mathcal{A}$.*

- *$\mathcal{A}$ makes a sequence of* ***signing requests*** *to the challenger. Each such request is a message m, which the challenger signs using sk, giving the resulting signature σ to $\mathcal{A}$.*

- *At the end of the game, $\mathcal{A}$ outputs (m^*, σ^*).*

- *We say $\mathcal{A}$* ***wins the game*** *if σ^* is a valid signature on m^* under pk, and m^* was not submitted as a signing request.*

Definition 2 (CMA security in GGM). *If $\mathcal{S}$ is based on computations in a certain group, we can also model such a CMA attack in the* ***generic group model****, in which all computations in the group done by $\mathcal{A}$ and the challenger are performed using the group oracle as described in Section 2. In this case, $\mathcal{A}$'s advantage in the corresponding CMA attack game is denoted* $\mathrm{CMA}^{\mathrm{ggm}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$.

4.1.1 Strong unforgeability

In some settings, the stronger security notion known as **strong unforgeability** is desirable. This notion of security means that not only is it hard to construct a valid signature on a message for which no signing request was issued, it is also hard to construct a valid signature on a message other than one that is *identical* to the response to some signing request on that same message.

It is well known that ECDSA is trivially not strongly unforgeable. Indeed, given a valid signature (s, t) on a message m, then $(-s, t)$ is also a valid signature on m. To see this, suppose $s\mathcal{R} = h\mathcal{G} + t\mathcal{D}$, where $h = \mathit{Hash}(m)$ and $t = \bar{C}(\mathcal{R})$. Then $\bar{C}(R) = \bar{C}(-R)$ and $(-s)(-\mathcal{R}) = h\mathcal{G} + t\mathcal{D}$. We say that the signatures (s, t) and $(-s, t)$ are **equivalent up to sign**, and we we say that ECDSA is **strongly unforgeable up to sign** if it is hard to construct a valid signature on a message other than one that is *equivalent up to sign* to the response to some signing request on that same message. As we shall observe, *all of our security proofs for the unforgeability of ECDSA and all of its variants also prove strong unforgeability up to sign.*

Strong unforgeability up to sign implies that ECDSA can be trivially converted to a strongly secure signature scheme, simply by requiring that a valid signature (s, t) satisfies $s \in \{[a]_q : 1 \leq a < q/2\}$, where $[\cdot]_q$ is the natural map from $\mathbb{Z}$ to $\mathbb{Z}_q$, and by having the signature algorithm choose the unique valid signature among $(\pm s, t)$.

4.2 Hash functions

Definition 3 (Random-preimage resistance). *Let Hash be a hash function whose output space is $\mathbb{Z}_q$. Let $\mathcal{A}$ be an adversary. We define* $\mathrm{RPR}\mathsf{adv}[\mathcal{A}, \mathit{Hash}]$ *to be the advantage of $\mathcal{A}$ in breaking the **random-preimage resistance** of Hash. This is defined as the probability that $\mathcal{A}$ wins the following game.*

- *The challenger chooses $h \in \mathbb{Z}_q$ uniformly at random and gives h to $\mathcal{A}$.*
- *$\mathcal{A}$ outputs m.*
- *We say $\mathcal{A}$ **wins the game** if $\mathit{Hash}(m) = h$.*

Definition 4 (Zero-preimage resistance). *Let Hash be a hash function whose output space is $\mathbb{Z}_q$. Let $\mathcal{A}$ be an adversary. We define* $\mathrm{ZPR}\mathsf{adv}[\mathcal{A}, \mathit{Hash}]$ *to be the advantage of $\mathcal{A}$ in breaking the **zero-preimage resistance** of Hash. This is defined as the probability that $\mathcal{A}$ wins the following game.*

- *$\mathcal{A}$ outputs m.*
- *We say $\mathcal{A}$ **wins the game** if $\mathit{Hash}(m) = 0$.*

Note that the probability of winning in this game is taken over the random choices of $\mathcal{A}$ as well as any random choices made in generating system parameters that define Hash.

Definition 5 (Collision resistance). *Let Hash be a hash function. Let $\mathcal{A}$ be an adversary. We define* $\mathrm{CR}\mathsf{adv}[\mathcal{A}, \mathit{Hash}]$ *to be the advantage of $\mathcal{A}$ in breaking the **collision resistance** of Hash. This is defined as the probability that $\mathcal{A}$ wins the following game.*

- *$\mathcal{A}$ outputs m, m'.*
- *We say $\mathcal{A}$ **wins the game** if $\mathit{Hash}(m) = \mathit{Hash}(m')$ but $m \neq m'$.*

Note that the probability of winning in this game is taken over the random choices of $\mathcal{A}$ as well as any random choices made in generating system parameters that define Hash.

5 Proof of security of ECDSA in the EC-GGM

In the EC-GGM model, the generator $\mathcal{G}$ is encoded as $\pi(1)$ and the public key $\mathcal{D}$ is encoded as $\pi(d)$ for randomly chosen $d \in \mathbb{Z}_q^*$. We assume that $d \neq 0$. These encodings of $\mathcal{G}$ and $\mathcal{D}$ are given to the adversary at the start of the signing attack game.

The adversary then interacts makes a sequence of queries to both the group and signing oracles. The signing oracle on a message m itself works as usual, computing

$$h = \mathit{Hash}(m),$$

but it uses the group oracle to compute the encoding of

$$\mathcal{R} = r\mathcal{G}.$$

```
1. Initialization:
   (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
   (b) $d \xleftarrow{\$} \mathbb{Z}_q^*$
   (c) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
   (d) invoke $(\texttt{map}, d)$ to obtain $\mathcal{D}$
   (e) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
   (a) if $i \notin \mathit{Domain}(\pi)$:
       i. $\mathcal{P} \xleftarrow{\$} E^*$;
          while $\mathcal{P} \in \mathit{Range}(\pi)$ do: $\mathcal{P} \xleftarrow{\$} E^*$
       ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
   (b) return $\pi(i)$
3. To process a group oracle query $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2)$:
   (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
       i. $i \xleftarrow{\$} \mathbb{Z}_q^*$;
          while $i \in \mathit{Domain}(\pi)$ do: $i \xleftarrow{\$} \mathbb{Z}_q^*$
       ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
   (b) invoke $(\texttt{map}, \pi^{-1}(\mathcal{P}_1)+\pi^{-1}(\mathcal{P}_2))$ and return the result
4. To process a request to sign $m$:
   (a) $h \leftarrow \mathit{Hash}(m) \in \mathbb{Z}_q$
   (b) $r \xleftarrow{\$} \mathbb{Z}_q^*$
   (c) invoke $(\texttt{map}, r)$ to get $\mathcal{R}$
   (d) $t \leftarrow \bar{C}(\mathcal{R}) \in \mathbb{Z}_q$
   (e) if $t = 0$ then return *fail*
   (f) if $h + td = 0$ then return *fail*
   (g) $s \leftarrow r^{-1}(h + td)$
   (h) return $(\mathcal{R}, s, t)$
```

Figure 2: Lazy-Sim

Note that we have

$$\mathcal{R} = s^{-1}h\mathcal{G} + s^{-1}t\mathcal{D},$$

where (s, t) is the signature. For simplicity, let us assume that $\mathcal{R}$ is output by the signing oracle as well.

At the end of the signing attack game, the adversary outputs a forgery (s^*, t^*) on a message m^*. The signature is then verified using the verification algorithm, computing

$$h^* = \mathit{Hash}(m^*),$$

and then again making use of the group oracle to compute the encoding of

$$\mathcal{R}^* = (s^*)^{-1}h^*\mathcal{G} + (s^*)^{-1}t^*\mathcal{D}.$$

We define three types of forgers.

Type I. $\mathcal{R}^* = \pm\mathcal{R}$ for some $\mathcal{R}$ computed by the signing oracle.

Type II. $\mathcal{R}^* \neq \pm\mathcal{R}$ for any $\mathcal{R}$ computed by the signing oracle, and $h^* \neq 0$.

Type III. Neither Type I or Type II.

5.1 A lazy simulation of the signature attack game

Instead of choosing the encoding function π at random at the beginning of the attack game, we can lazily construct π a bit at a time. That is, we represent π as a set of pairs $(i, \mathcal{P})$ which grows over time — such a pair $(i, \mathcal{P})$ represents the relation $\pi(i) = \mathcal{P}$. Here, we give the entire logic for both the group and signing oracles in the forgery attack game. Figure 2 gives the details of **Lazy-Sim**.

At the end of the attack game, the adversary will output a forgery (s^*, t^*) on a message m^*. The verification routine will be used to verify this signature, and this will use the

```
1. Initialization:
   (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
   (b) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
   (c) invoke $(\texttt{map}, \texttt{D})$ to obtain $\mathcal{D}$
   (d) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
   (a) if $i \notin \mathit{Domain}(\pi)$:
       i. $\mathcal{P} \overset{\$}{\leftarrow} E^*$;
          if $\mathcal{P} \in \mathit{Range}(\pi)$ then abort
       ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
   (b) return $\pi(i)$
3. To process a group oracle query $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2)$:
   (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
       i. $i \overset{\$}{\leftarrow} \mathbb{Z}_q^*$;
          if $i \in \mathit{Domain}(\pi)$ then abort
       ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
   (b) invoke $(\texttt{map}, \pi^{-1}(\mathcal{P}_1) + \pi^{-1}(\mathcal{P}_2))$ and return the result
4. To process a request to sign $m$:
   (a) $h \leftarrow \mathit{Hash}(m) \in \mathbb{Z}_q$
   (b) $\mathcal{R} \overset{\$}{\leftarrow} E^*$
   (c) if $\mathcal{R} \in \mathit{Range}(\pi)$ then abort
   (d) $t \leftarrow \bar{C}(\mathcal{R}) \in \mathbb{Z}_q$
   (e) if $t = 0$ then abort
   (f) $s \overset{\$}{\leftarrow} \mathbb{Z}_q^*$
   (g) $r \leftarrow s^{-1}(h + t\texttt{D})$
   (h) if $r \in \mathit{Domain}(\pi)$ then abort
   (i) add $(-r, -\mathcal{R})$ and $(r, \mathcal{R})$ to $\pi$
   (j) return $(\mathcal{R}, s, t)$
```

Figure 3: Symbolic-Sim

`add` queries to perform the computation, which will take $O(\log q)$ group oracle queries. We denote by N_{grp} the total number of group oracle queries explicitly made by the adversary, with the understanding that this includes the group oracle queries used to verify the the forgery, as well as the group oracle queries used to generate $\mathcal{G}$ and $\mathcal{D}$, but not including group oracle queries used in the signing queries. We let N_{sig} denote the number of signing queries made by the adversary, and set $N := N_{\text{sig}} + N_{\text{grp}}$.

This lazy simulation is perfectly faithful. Specifically, the advantage of any adversary in the signature attack game using this lazy simulation of the group oracle is identical to that using the group oracle as originally defined.

5.2 A symbolic simulation of the signature attack game

We now define a **symbolic** simulation of the attack game. The essential difference in this game is that *Domain*(π) will now consist of polynomials of the form $a + b\texttt{D}$, where $a, b \in \mathbb{Z}_q$ and $\texttt{D}$ is an indeterminant. Here, $\texttt{D}$ symbolically represents the value of d. Note that π will otherwise still satisfy all of the requirements of an encoding function. Figure 3 gives the details of **Symbolic-Sym**.

Lemma 1. *The difference between the adversary's forging advantage in the Lazy-Sim and Symbolic-Sim games (as described in Sections 5.1 and 5.2) is $O(N^2/q)$.*

Proof. See Appendix A. □

5.3 Security analysis of ECDSA against a symbolic simulator

We now analyze the security of ECDSA in the EC-GGM. We assume here that the attack is taking place with respect to the symbolic simulator (see Section 5.2).

Type I forger. Now consider a Type I forger, where $\mathcal{R}^* = \pm\mathcal{R}$ for some $\mathcal{R}$ produced by the signing oracle (which must be unique). This means

$$(s^*)^{-1}(h^* + t^*\mathtt{D}) = \pm s^{-1}(h + t\mathtt{D}) \quad \text{and} \quad t^* = t.$$

In other words, for $\eta \in \{\pm 1\}$, we have

$$(s^*)^{-1}(h^* + t\mathtt{D}) = \eta s^{-1}(h + t\mathtt{D}),$$

which gives us the two equations

$$(s^*)^{-1}h^* = \eta s^{-1}h \quad \text{and} \quad (s^*)^{-1}t = \eta s^{-1}t.$$

These two equations imply $h^* = h$, which implies a collision on the hash function *Hash*.

So we have shown: if $\mathcal{A}$ is an efficient adversary that produces a Type I forgery with probability ϵ_{I}, then there is an efficient adversary $\mathcal{B}_{\mathrm{I}}$ such that

$$\mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{I}}, \mathit{Hash}] \geq \epsilon_{\mathrm{I}}.$$

The running time of $\mathcal{B}_{\mathrm{I}}$ is essentially the same as that of $\mathcal{A}$.

Type II forger. Now consider a Type II forger, where $\mathcal{R}^* \neq \pm\mathcal{R}$ for any $\mathcal{R}$ produced by the signing oracle. Suppose

$$\pi^{-1}(\mathcal{R}^*) = a + b\mathtt{D}.$$

By the verification equation, we also have

$$\pi^{-1}(\mathcal{R}^*) = (s^*)^{-1}(h^* + t^*\mathtt{D}).$$

Thus, we have

$$a = (s^*)^{-1}h^* \quad \text{and} \quad b = (s^*)^{-1}t^*.$$

These identities, along with the assumption that $h^* \neq 0$, imply that $b \neq 0$, $a \neq 0$, and

$$t^* = h^* a^{-1} b.$$

The group element $\mathcal{R}^*$ must have been generated at random by some group oracle query made directly by the adversary (this follows from the fact that $b \neq 0$). Since the coefficients a, b were already determined before this query, it follows that the value of $\mathcal{R}^*$ is independent of these coefficients.

We want to use this Type II forger to break the random-preimage resistance of *Hash*. That is, we are given random $h^\dagger \in \mathbb{Z}_q$ and want to find a preimage of $h^\dagger$ under *Hash*. To do this, we will **guess the group oracle query** that will produce the value $\mathcal{R}^*$ in the forgery, and then we will **run our sampling algorithm** to compute

$$t^\dagger \leftarrow h^\dagger a^{-1} b, \quad \mathcal{R}^\dagger \xleftarrow{\$} \mathit{Samp}(t^\dagger).$$

If the sampler fails, then we abort. Otherwise, we set $\mathcal{R}^* := \mathcal{R}^\dagger$ and $t^* := t^\dagger$ and proceed as usual: if the adversary forges a signature, we succeed in finding a preimage of $h^\dagger$.

So we have shown: if $\mathcal{A}$ is an efficient adversary that makes at most N signing or group queries, and which produces a Type II forgery with probability ϵ_{II}, then there is an efficient adversary $\mathcal{B}_{\mathrm{II}}$ such that

$$\mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{II}}, \mathit{Hash}] \geq \frac{1/4 + o(1)}{N}\epsilon_{\mathrm{II}}.$$

The running time of $\mathcal{B}_{\mathrm{II}}$ is essentially the same as that of $\mathcal{A}$.

Type III forger. A Type III forger produces a forgery with $h^* = 0$. We rule this out by simply assuming that it is hard to find a preimage of 0 under *Hash*.

So we have shown: if $\mathcal{A}$ is an efficient adversary that produces a Type III forgery with probability $\epsilon_{\rm III}$, then there is an efficient adversary $\mathcal{B}_{\rm III}$ such that

$$\mathrm{ZPR}\mathsf{adv}[\mathcal{B}_{\rm III}, \mathit{Hash}] \geq \epsilon_{\rm III}.$$

The running time of $\mathcal{B}_{\rm III}$ is essentially the same as that of $\mathcal{A}$.

Putting this all together with Lemma 1, we obtain:

Theorem 1. *Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\rm ecdsa}$ as in Definition 2 that makes at most N signing or group queries. Then there exist adversaries $\mathcal{B}_{\rm I}$, $\mathcal{B}_{\rm II}$, and $\mathcal{B}_{\rm III}$, whose running times are essentially the same as $\mathcal{A}$, such that*

$$\begin{aligned}\mathrm{CMA}^{\rm ggm}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\rm ecdsa}] \leq\ & \mathrm{CR}\mathsf{adv}[\mathcal{B}_{\rm I}, \mathit{Hash}] + \\ & (4 + o(1))N \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\rm II}, \mathit{Hash}] + \\ & \mathrm{ZPR}\mathsf{adv}[\mathcal{B}_{\rm III}, \mathit{Hash}] + \\ & O(N^2/q).\end{aligned}$$

Proof. Applying Lemma 1, we have

$$\mathrm{CMA}^{\rm ggm}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\rm ecdsa}] \leq \epsilon_{\rm I} + \epsilon_{\rm II} + \epsilon_{\rm III} + O(N^2/q).$$

The rest follows from the arguments made above. □

NOTES:

1. All three assumptions we make — collision resistance, random-preimage resistance, and zero-preimage resistance — are necessary conditions, in the sense that it is trivial to break the scheme if any of them are false.

2. The above analysis shows that ECDSA is secure under the same assumptions, even if we give the adversary access to a "raw" signing oracle, where the input is h, not m. Of course, in this model, the notion of a forgery must be modified appropriately, to disallow forgery on any message m^* for which $H(m^*)$ was submitted as a "raw" signing query.

3. It is easy to see that the result of Theorem 1 applies equally well to *strong unforgeability up to sign* (see Section 4.1.1). Indeed, the only place in the proof where we explicitly used the assumption that m^* was not submitted as a signing request was in the analysis of a Type I forgery, where $\mathcal{R}^* = \pm\mathcal{R}$ for some $\mathcal{R}$ produced by the signing oracle on input m. We then used the fact that $h^* = h$ but $m^* \neq m$ to break the collision resistance of *Hash*. However, if $m^* = m$, then the fact that $\mathcal{R}^* = \pm\mathcal{R}$ implies that (s^*, t^*) is equivalent up to sign to the signature (s, t) produced by the signing oracle. Therefore, the signature (s^*, t^*) is not a forgery in the sense of strong unforgeability up to sign.

6 ECDSA with additive key derivation

We assume that the secret key $d \in \mathbb{Z}_p$ is used as a master key to derive secret subkeys of the form $d + e$ for a "tweak" $e \in \mathbb{Z}_q$. For such a derived secret subkey, we can compute the corresponding derived public subkey from the public key $\mathcal{D}$ as $\mathcal{D} + e\mathcal{G}$.

As we will see, it is impossible to achieve security without some restriction on the choice of tweaks. We assume that any tweak must come from a set $\mathfrak{E} \subseteq \mathbb{Z}_q$ of allowed tweaks that is chosen before the attack game starts. This can be enforced in several ways, one of which is to obtain tweaks as the output of a hash function which is modeled as a random oracle. In Appendix D we provide an analysis of the BIP32 key derivation function, which justifies modeling it as a (public use) random oracle. As we will see, security will degrade linearly in $|\mathfrak{E}|$. In Section 6.1, we provide an alternative analysis in terms of concrete security properties of the hash function used to derive tweaks.

The CMA security game in Definition 1 (as well as Definition 2) is modified so that the signing oracle takes a message m and a tweak e. Similarly, the adversary must output a forgery on a specific message m^* under specific tweak e^*, and the forgery only counts if the pair (m^*, e^*) was not given to the signing oracle.

We define $\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{akd}}\mathsf{adv}[\mathcal{A}, \mathcal{S}, \mathfrak{E}]$ to be adversary $\mathcal{A}$'s advantage in winning this modified CMA game in the EC-GGM.

Lemma 1 is seen to hold as well in this setting, where to process a signing query (h, e), the symbolic simulator runs the same algorithm as before, but with $e + \mathtt{D}$ in place of $\mathtt{D}$.

We now analyze the security of ECDSA in this setting, assuming a symbolic simulator, just as we did in Section 5.3

Type I forger. Consider a Type I forger, where $\mathcal{R}^* = \pm\mathcal{R}$ for some $\mathcal{R}$ produced by the signing oracle (which must be unique). This means

$$(s^*)^{-1}(h^* + t^*(e^* + \mathtt{D})) = \pm s^{-1}(h + t(e + \mathtt{D})) \quad \text{and} \quad t^* = t.$$

In other words, for $\eta \in \{\pm 1\}$, we have

$$(s^*)^{-1}(h^* + te^* + t\mathtt{D}) = \eta s^{-1}(h + te + t\mathtt{D}),$$

which gives us the two equations

$$(s^*)^{-1}(h^* + te^*) = \eta s^{-1}(h + te) \quad \text{and} \quad (s^*)^{-1}t = \eta s^{-1}t.$$

These two equations imply

$$h^* + te^* = h + te. \tag{1}$$

If $e^* = e$, then we have $h^* = h$, and so we can use the forging adversary to break the collision resistance of *Hash* as before. Let us call this a **Type Ia forgery**.

Otherwise, we have

$$t = \frac{h^* - h}{e - e^*}.$$

Let us call this a **Type Ib forgery**. We want to use this Type Ib forger to break the random-preimage resistance of *Hash*. That is, we are given random $h^\dagger \in \mathbb{Z}_q$ and want to find a preimage of $h^\dagger$ under *Hash*. To do this, we will **guess the relevant signing query and the tweak** e^*. We then we will run our sampling algorithm to compute

$$t^\dagger \leftarrow \frac{h^\dagger - h}{e - e^*}, \quad \mathcal{R}^\dagger \overset{\$}{\leftarrow} \mathit{Samp}(t^\dagger).$$

If the sampler fails, then our forger fails. Otherwise, we set $\mathcal{R} := \mathcal{R}^\dagger$ and $t := t^\dagger$ and proceed as usual: if the adversary forges a signature, we succeed in finding a preimage of $h^\dagger$.

So we have shown:

- If $\mathcal{A}$ is an efficient adversary that produces a Type Ia forgery with probability ϵ_{Ia}, then there is an efficient adversary $\mathcal{B}_{\mathrm{Ia}}$ such that

  $$\mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{Ia}}, \mathit{Hash}] \geq \epsilon_{\mathrm{Ia}}.$$

  The running time of $\mathcal{B}_{\mathrm{Ia}}$ is essentially the same as that of $\mathcal{A}$.

- If $\mathcal{A}$ is an efficient adversary that makes at most N_{sig} signing queries, and which produces a Type Ib forgery with probability ϵ_{Ib}, then there is an efficient adversary $\mathcal{B}_{\mathrm{Ib}}$ such that finding advantage for *Hash* is at least

  $$\mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{Ib}}, \mathit{Hash}] \geq \frac{1/4 + o(1)}{N_{\mathrm{sig}}|\mathfrak{E}|}\epsilon_{\mathrm{Ib}}.$$

  The running time of $\mathcal{B}_{\mathrm{Ib}}$ is essentially the same as that of $\mathcal{A}$.

NOTES:

1. Security really does degrade as $|\mathfrak{E}|$ gets large. In particular, if $|\mathfrak{E}| = \Theta(q^{1/2})$, then for fixed h, t, and e, an adversary can expect to find $(h^*, e^*) \neq (h, e)$ satisfying (1) in time $O(q^{1/2})$, which is enough to forge a signature.

Type II forger. Now consider a Type II forger playing against our symbolic simulator, where $\mathcal{R}^* \neq \pm\mathcal{R}$ for any $\mathcal{R}$ produced by the signing oracle. Suppose

$$\pi^{-1}(\mathcal{R}^*) = a + b\mathtt{D}.$$

By the verification equation, we also have

$$\pi^{-1}(\mathcal{R}^*) = (s^*)^{-1}(h^* + t^*(e^* + \mathtt{D})).$$

Thus, we have

$$a = (s^*)^{-1}(h^* + t^*e^*) \quad \text{and} \quad b = (s^*)^{-1}t^*.$$

These identities, along with the assumption that $h^* \neq 0$, imply $b \neq 0$, $a - be^* \neq 0$, and

$$t^* = \frac{bh^*}{a - be^*}. \tag{2}$$

The group element $\mathcal{R}^*$ must have been generated at random by some group oracle query made directly by the adversary (this follows from the fact that $b \neq 0$). Since the coefficients

a, b were already determined before this query, it follows that the value of $\mathcal{R}^*$ is independent of these coefficients.

We want to use this Type II forger to break the random-preimage resistance of *Hash*. That is, we are given random $h^\dagger \in \mathbb{Z}_q$ and want to find a preimage of $h^\dagger$ under *Hash*. To do this, we will **guess the relevant group oracle query that will produce the value $\mathcal{R}^*$ in the forgery, as well as the tweak e^***. Then we will run our sampling algorithm to compute

$$t^\dagger \leftarrow \frac{bh^\dagger}{a - be^*}, \quad \mathcal{R}^\dagger \xleftarrow{\$} Samp(t^\dagger).$$

If the sampler fails, then our forger fails. Otherwise, we set $\mathcal{R}^* := \mathcal{R}^\dagger$ and $t^* := t^\dagger$ and proceed as usual: if the adversary forges a signature, we succeed in finding a preimage of $h^\dagger$.

So we have shown: if $\mathcal{A}$ is an efficient adversary that makes at most N signing or group queries, and which produces a Type II forgery with probability ϵ_{II}, then there is an efficient adversary $\mathcal{B}_{\text{II}}$ such that

$$\text{RPR}\mathsf{adv}[\mathcal{B}_{\text{II}}, \mathit{Hash}] \geq \frac{1/4 + o(1)}{N|\mathfrak{E}|}\epsilon_{\text{II}}.$$

The running time of $\mathcal{B}_{\text{II}}$ is essentially the same as that of $\mathcal{A}$.

NOTES:

1. Again, security really does degrade as $|\mathfrak{E}|$ gets large. In particular, if $|\mathfrak{E}| = \Theta(q^{1/2})$, then for fixed a, b, and t^*, an adversary can expect to find (h^*, e^*) satisfying (2) in time $O(q^{1/2})$, which is enough to forge a signature.

Type III forger. As above, a Type III forgery gives us a forgery with $h^* = 0$. We rule this out by simply assuming that it is hard to find a preimage of 0 under *Hash*.

Theorem 2. *Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\text{ecdsa}}$ as in Definition 2 with **additive key derivation** that makes at most N signing or group queries, of which N_{sig} are signing queries. Then there exist adversaries $\mathcal{B}_{\text{Ia}}$, $\mathcal{B}_{\text{Ib}}$, $\mathcal{B}_{\text{II}}$, and $\mathcal{B}_{\text{III}}$, whose running times are essentially the same as $\mathcal{A}$, such that*

$$\begin{aligned}\text{CMA}^{\text{ggm}}_{\text{akd}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{ecdsa}}, \mathfrak{E}] \leq\ & \text{CR}\mathsf{adv}[\mathcal{B}_{\text{Ia}}, \mathit{Hash}] + \\ & (4 + o(1))N_{\text{sig}}|\mathfrak{E}| \cdot \text{RPR}\mathsf{adv}[\mathcal{B}_{\text{Ib}}, \mathit{Hash}] + \\ & (4 + o(1))N|\mathfrak{E}| \cdot \text{RPR}\mathsf{adv}[\mathcal{B}_{\text{II}}, \mathit{Hash}] + \\ & \text{ZPR}\mathsf{adv}[\mathcal{B}_{\text{III}}, \mathit{Hash}] + \\ & O(N^2/q).\end{aligned}$$

Proof. Applying Lemma 1, adapted to additive key derivation, we have

$$\text{CMA}^{\text{ggm}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{ecdsa}}] \leq \epsilon_{\text{Ia}} + \epsilon_{\text{Ib}} + \epsilon_{\text{II}} + \epsilon_{\text{III}} + O(N^2/q).$$

The rest follows from the arguments made above. □

NOTES:

1. This analysis also shows that ECDSA with additive key derivation is secure under the same assumptions, even if we give the adversary access to a "raw" signing oracle, where the input is h, not m. It even remains secure if the signing tweak e is not constrained to lie in the set $\mathfrak{E}$. It is really only the forging tweak e^* that must be constrained.

2. It is easy to see that the result of Theorem 2 applies equally well to *strong unforgeability up to sign* (see Section 4.1.1). Indeed, the only place in the proof where we explicitly used the assumption that (m^*, e^*) was not submitted as a signing request was in the analysis of a Type Ia forgery, where $\mathcal{R}^* = \pm\mathcal{R}$ for some $\mathcal{R}$ produced by the signing oracle on input (m, e), where $e^* = e$. Just as we noted in the discussion after Theorem 1, either $m^* = m$ and (s^*, t^*) is not a forgery in the sense of strong unforgeability up to sign, or $m^* \neq m$ and we can break the collision resistance of *Hash*.

6.1 Alternative analysis

Suppose that a tweak $e \in \mathbb{Z}_q$ is derived from an identifier via the hash function Hash' as

$$e \leftarrow \mathit{Hash}'(\mathit{id}),$$

and there is otherwise no restriction on the set of valid tweaks. Here, Hash' could be the same as *Hash*, or it could be a different hash function (for example, BIP32 uses a specific hashing mechanism that is also parameterized by the verification key $\mathcal{D}$). In this setting, the definition of security would need to be adjusted so that the rules for valid forgeries are defined in terms of identities rather than tweaks.

In this setting, the term

$$\mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{Ia}}, \mathit{Hash}]$$

appearing in Theorem 2 would have to be replaced by

$$\mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{Ia}}, \mathit{Hash}] + \mathrm{CR}\mathsf{adv}[\mathcal{B}'_{\mathrm{Ia}}, \mathit{Hash}'].$$

Also in this setting, the term

$$|\mathfrak{E}| \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{Ib}}, \mathit{Hash}]$$

appearing in Theorem 2 can be replaced by

$$2\mathrm{sum}\mathsf{adv}[\mathcal{B}_{\mathrm{Ib}}, \mathit{Hash}, \mathit{Hash}'],$$

where $2\mathrm{sum}\mathsf{adv}$ is defined as follows:

Definition 6 (2sum intractability). *Let $\mathit{Hash}, \mathit{Hash}1$ be a hash function whose output space is $\mathbb{Z}_q$. Let $\mathcal{A}$ be an adversary. We define $2\mathrm{sum}\mathsf{adv}[\mathcal{A}, \mathit{Hash}, \mathit{Hash}']$ to be the advantage of $\mathcal{A}$ in breaking the **2sum property** of Hash and Hash′. This is defined as the probability that $\mathcal{A}$ wins the following game.*

- *Adversary chooses $h, e \in \mathbb{Z}_q$ and sends gives these to the challenger.*
- *The challenger chooses $t \in \mathbb{Z}_q$ uniformly at random and gives t to $\mathcal{A}$.*
- *$\mathcal{A}$ outputs m^*, id^*.*
- *We say $\mathcal{A}$ **wins the game** if*

  $$h + te = h^* + te^*,$$

  where $e \neq e^*$, $h^* := Hash(m^*)$, and $e^* := Hash'(id^*)$.

Also in this setting, the term

$$|\mathfrak{C}| \cdot \text{RPR}\mathsf{adv}[\mathcal{B}_{\text{II}}, Hash]$$

appearing in Theorem 2 can be replaced by

$$2\text{sum}\mathsf{adv}[\mathcal{B}_{\text{II}}, Hash, Hash'].$$

The value e in Definition 6 would be set to the value a/b in the proof of Theorem 2, the value h in the definition would be set to 0, and the value t in the definition would correspond to the value t^* in the proof.

Note that we could weaken the definition of 2sum intractability by having the adversary supply preimages of h and e. Under this weaker definition, we could no longer prove security with respect to raw signing queries under this assumption, however.

We leave it as an easy exercise for the reader to show that if $Hash$ and $Hash'$ are modeled as random oracles, then

$$2\text{sum}^{\text{ro}}\mathsf{adv}[\mathcal{A}, Hash, Hash'] \leq \frac{QQ'}{q},$$

where $2\text{sum}^{\text{ro}}\mathsf{adv}[\mathcal{A}, Hash, Hash']$ is the corresponding advantage in the random oracle model, Q is a bound on the number of queries to $Hash$ made by $\mathcal{A}$, and Q' is a bound on the number of queries to $Hash'$ made by $\mathcal{A}$.

7 ECDSA with presignatures

In some settings, it is convenient to precompute various pairs

$$(r, \mathcal{R})$$

where $r \xleftarrow{\$} \mathbb{Z}_q^*$ and $\mathcal{R} \leftarrow r\mathcal{G}$. When processing a request to sign a message, we can allocate one such precomputed pair and use it to finish the computation of the signature. So long as neither $\mathcal{R}$ is not revealed to the adversary before he makes a signing query, our proof of security goes through unchanged. However, there are optimizations in some settings (especially in threshold signing protocols) that can be exploited if we do in fact reveal $\mathcal{R}$ to the adversary before he chooses which message to sign using the value of $\mathcal{R}$.

In the forgery game, we allow the adversary to make `presig` queries, which generate a pair $(r, \mathcal{R})$ as above. In a signing request, the adversary also specifies an index k to specify that the kth presignature should be used to sign the given message. The adversary is not allowed to specify the same presignature index for two distinct signing requests.

```
1. Initialization:
   (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
   (b) $d \xleftarrow{\$} \mathbb{Z}_q^*$
   (c) invoke $(\mathtt{map}, 1)$ to obtain $\mathcal{G}$
   (d) invoke $(\mathtt{map}, d)$ to obtain $\mathcal{D}$
   (e) $k \leftarrow 0$; $K \leftarrow \emptyset$
   (f) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\mathtt{map}, i)$:
   (a) if $i \notin \mathit{Domain}(\pi)$:
         i. $\mathcal{P} \xleftarrow{\$} E^*$;
            while $\mathcal{P} \in \mathit{Range}(\pi)$ do: $\mathcal{P} \xleftarrow{\$} E^*$
        ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
   (b) return $\pi(i)$
3. To process a group oracle query $(\mathtt{add}, \mathcal{P}_1, \mathcal{P}_2)$:
   (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
         i. $i \xleftarrow{\$} \mathbb{Z}_q^*$;
            while $i \in \mathit{Domain}(\pi)$ do: $i \xleftarrow{\$} \mathbb{Z}_q^*$
        ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
   (b) invoke $(\mathtt{map}, \pi^{-1}(\mathcal{P}_1) + \pi^{-1}(\mathcal{P}_2))$ and return the result
4. To process a presignature request:
   (a) $k \leftarrow k + 1$
   (b) $r_k \xleftarrow{\$} \mathbb{Z}_q^*$
   (c) invoke $(\mathtt{map}, r_k)$ to get $\mathcal{R}_k$
   (d) $t_k \leftarrow \bar{C}(\mathcal{R}_k) \in \mathbb{Z}_q$
   (e) if $t_k = 0$ then return *fail*
   (f) $K \leftarrow K \cup \{k\}$;  return $\mathcal{R}_k$
5. To process a request to sign $m_k$ using presignature number $k \in K$:
   (a) $K \leftarrow K \setminus \{k\}$
   (b) $h_k \leftarrow \mathit{Hash}(m_k) \in \mathbb{Z}_q$
   (c) if $h_k + t_k d = 0$ then return *fail*
   (d) $s_k \leftarrow r_k^{-1}(h_k + t_k d)$
   (e) return $(s_k, t_k)$
```

Figure 4: Lazy-Sim (with presignatures)

7.1 A lazy simulation of the signature attack game

We start with the analog of Lazy-Sim in Section 5.1, but now with presignatures. Figure 4 gives the details of **Lazy-Sim**.

7.2 A symbolic simulation of the signature attack game

We now define a **symbolic** simulation of the attack game, which is the analog of Symbolic-Sim in Section 5.2. In this setting, however, *Domain*(π) will now consist of polynomials of the form

$$a + b\mathtt{D} + c_1\mathtt{R}_1 + c_2\mathtt{R}_2 + \cdots ,$$

where $a, b, c_1, c_2, \ldots \in \mathbb{Z}_q$, and $\mathtt{D}, \mathtt{R}_1, \mathtt{R}_2, \ldots$ are indeterminants. Here, $\mathtt{D}$ symbolically represents the value of d, and $\mathtt{R}_k$ symbolically represents the value of r_k. Figure 5 gives the details of **Symbolic-Sim**.

Lemma 2. *The difference between the adversary's forging advantage in the Lazy-Sim and Symbolic-Sim games (as described in Sections 7.1 and 7.2) is $O(N^2/q)$.*

Proof. See Appendix B. □

Since our symbolic simulation is used in our reductions to various hardness assumptions about *Hash*, we have to take into account the extra cost associated with computing with polynomials in the variables $\mathtt{D}, \mathtt{R}_1, \mathtt{R}_2, \ldots$. Let U denote the maximum number of *unused* presignatures at any point in time, i.e., the maximum size of the set K attained throughout the game. Assuming we use hash tables as appropriate, the symbolic simulation can be

```
1. Initialization:
   (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
   (b) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
   (c) invoke $(\texttt{map}, \texttt{D})$ to obtain $\mathcal{D}$
   (d) $k \leftarrow 0$; $K \leftarrow \emptyset$
   (e) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
   (a) if $i \notin \mathit{Domain}(\pi)$:
       i. $\mathcal{P} \xleftarrow{\$} E^*$;
          if $\mathcal{P} \in \mathit{Range}(\pi)$ then abort
       ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
   (b) return $\pi(i)$
3. To process a group oracle query $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2)$:
   (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
       i. $i \xleftarrow{\$} \mathbb{Z}_q^*$;
          if $i \in \mathit{Domain}(\pi)$ then abort
       ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
   (b) invoke $(\texttt{map}, \pi^{-1}(\mathcal{P}_1) + \pi^{-1}(\mathcal{P}_2))$ and return the result
4. To process a presignature request:
   (a) $k \leftarrow k + 1$
   (b) invoke $(\texttt{map}, \texttt{R}_k)$ to get $\mathcal{R}_k$
   (c) $t_k \leftarrow \bar{C}(\mathcal{R}_k) \in \mathbb{Z}_q$
   (d) if $t_k = 0$ then abort
   (e) $K \leftarrow K \cup \{k\}$; return $\mathcal{R}_k$
5. To process a request to sign $m_k$ using presignature number $k \in K$:
   (a) $K \leftarrow K \setminus \{k\}$
   (b) $h_k \leftarrow \mathit{Hash}(m_k) \in \mathbb{Z}_q$
   (c) $s_k \xleftarrow{\$} \mathbb{Z}_q^*$
   (d) substitute $s_k^{-1}(h_k + t_k\texttt{D})$ for $\texttt{R}_k$ throughout $\mathit{Domain}(\pi)$, and abort if any two polynomials collapse
   (e) return $(s_k, t_k)$
```

Figure 5: Symbolic-Sim (with presignatures)

implemented so as to have an expected running time that is $O(UN)$ (with good tail bounds on the running time as well). This degradation in the running time by a factor of U for the extra bookkeeping seems unavoidable. If one views *Hash* as a random oracle, then this degradation plays no role, as then we have a perfectly information-theoretic result.

7.3 Security of ECDSA with presignatures

The results proved in Section 5.3 on basic ECDSA (without key derivation) do not carry through without modification. To analyze security in the setting, we need a new assumption on *Hash*:

Definition 7 (Ratio resistance). *Let Hash be a hash function whose output space is $\mathbb{Z}_q$. Let $\mathcal{A}$ be an adversary. We define* $\mathrm{RR}\mathsf{adv}[\mathcal{A}, \mathit{Hash}]$ *to be the advantage of $\mathcal{A}$ in breaking the **ratio resistance** of Hash. This is defined as the probability that $\mathcal{A}$ wins the following game.*

- *The challenger chooses $\rho \in \mathbb{Z}_q^*$ uniformly at random and gives ρ to $\mathcal{A}$.*
- *$\mathcal{A}$ outputs messages m and m^*.*
- *We say $\mathcal{A}$ **wins the game** if $\mathit{Hash}(m^*) \neq 0$ and $\mathit{Hash}(m)/\mathit{Hash}(m^*) = \rho$.*

If we view *Hash* as a random oracle, then the best type of ratio resistance attack is a birthday attack.

We define $\mathrm{CMA}_{\mathrm{ps}}^{\mathrm{ggm}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ to be adversary $\mathcal{A}$'s advantage in winning the CMA game with presignatures in the EC-GGM.

Theorem 1 then becomes:

Theorem 3. *Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\mathrm{ecdsa}}$ as in Definition 2 with **presignatures** that makes at most N presignature, signing, or group queries. Let U denote the maximum number of* unused *presignatures at any point in time. Then there exist adversaries $\mathcal{B}_{\mathrm{I}}$, $\mathcal{B}_{\mathrm{IIa}}$, $\mathcal{B}_{\mathrm{IIb}}$, $\mathcal{B}_{\mathrm{IIc}}$, and $\mathcal{B}_{\mathrm{III}}$, whose running times are essentially the same as $\mathcal{A}$ plus $O(UN)$, such that*

$$\begin{aligned}\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{ps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{ecdsa}}] \leq {} & \mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{I}}, \mathit{Hash}] + \\ & (4 + o(1))N \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIa}}, \mathit{Hash}] + \\ & (4 + o(1))N \cdot \mathrm{RR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIb}}, \mathit{Hash}] + \\ & UN \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIc}}, \mathit{Hash}] + \\ & \mathrm{ZPR}\mathsf{adv}[\mathcal{B}_{\mathrm{III}}, \mathit{Hash}] + \\ & O(N^2/q).\end{aligned}$$

Proof. Everything goes through as in the proof of Theorem 1, except for the analysis of Type II forgeries.

Consider the point in time when the adversary queries the group oracle to obtain $\mathcal{R}^*$ for the first time. Let us call this a **Type IIa** forgery if at this time, $\pi^{-1}(\mathcal{R}^*)$ is of the form $a + b\mathtt{D}$. Type IIa forgeries can be dealt with in exactly the same way as Type II forgeries in the proof of Theorem 1.

Now, consider a Type II forgery that is not a Type IIa forgery. For such a forgery, the initial preimage of $\mathcal{R}^*$ is a polynomial that involves the indeterminants $\mathtt{R}_1, \mathtt{R}_2, \ldots$. However, before the attack ends, all of these variables must be substituted via signing queries — indeed, if the attack ends with a forgery, we must have $\pi^{-1}(\mathcal{R}^*) = (s^*)^{-1}(h^* + t^*\mathtt{D})$.

Renaming variables as necessary, suppose that at the time $\mathcal{R}^*$ is initially generated, we have

$$\pi^{-1}(\mathcal{R}^*) = a + b\mathtt{D} + c_1\mathtt{R}_1 + \cdots + c_\ell\mathtt{R}_\ell,$$

where the c_i's are nonzero, and that during the attack, we substitute

$$\mathtt{R}_i \mapsto s_i^{-1}(h_i + t_i\mathtt{D}) \quad \text{for } i = 1, \ldots, \ell$$

in that order. Let us define a **Type IIb forgery** to be one with

$$\frac{h_1}{t_1} = \cdots = \frac{h_\ell}{t_\ell} = \frac{h^*}{t^*}, \tag{3}$$

and we define a **Type IIc forgery** to be a Type II forgery that is neither Type IIa or IIb.

We can use a Type IIb forger to break the ratio resistance of *Hash*. Note that the initial preimage of $\mathcal{R}^*$ cannot be of the form $\pm\mathtt{R}_k$, as otherwise this would be a Type I forgery; in particular, the group element $\mathcal{R}^*$ must be generated at random via a group oracle query made directly by the adversary. Therefore, given the ratio-resistance challenge ρ, we **guess the group oracle query that produces** $\mathcal{R}^*$, pick one of the variables $\mathtt{R}_i$ arbitrarily from among the variables $\mathtt{R}_1, \mathtt{R}_2, \ldots, \mathtt{R}_\ell$ appearing in $\pi^{-1}(\mathcal{R}^*)$ at that time $\mathcal{R}^*$ is generated, and run the sampler on input $t^* = t_i/\rho$ to generate $\mathcal{R}^*$. This is the adversary $\mathcal{B}_{\mathrm{IIb}}$ in Theorem 3. Note that adversary $\mathcal{B}_{\mathrm{IIb}}$ will succeed if its guess at $\mathcal{R}^*$ was correct, regardless of which of the variables $\mathtt{R}_i$ it chooses.

We can use a Type IIc forger to break the random-preimage resistance of *Hash*. This is the adversary $\mathcal{B}_{\text{IIc}}$ in Theorem 3. To understand the design of adversary $\mathcal{B}_{\text{IIc}}$, consider a Type IIc forgery. For $i = 0, \ldots, \ell$, define

$$A_i \coloneqq a + \sum_{j \leq i} c_j h_j / s_j \quad \text{and} \quad B_i \coloneqq b + \sum_{j \leq i} c_j t_j / s_j.$$

At the end of the attack, we must have

$$\pi^{-1}(\mathcal{R}^*) = A_\ell + B_\ell \mathtt{D},$$

and so the forgery must satisfy:

$$A_\ell = (s^*)^{-1} h^* \quad \text{and} \quad B_\ell = (s^*)^{-1} t^*. \tag{4}$$

These two equations imply

$$A_\ell = B_\ell \cdot h^*/t^*. \tag{5}$$

Using the fact that $A_\ell = A_{\ell-1} + c_\ell h_\ell / s_\ell$ and $B_\ell = B_{\ell-1} + c_\ell t_\ell / s_\ell$, we can rewrite (5) as

$$(A_{\ell-1} - B_{\ell-1} h_\ell / t_\ell) = \underbrace{(B_{\ell-1} + s_\ell^{-1} c_\ell t_\ell)}_{=B_\ell} (h^*/t^* - h_\ell / t_\ell). \tag{6}$$

From (6), it is clear that either

(a) $A_{\ell-1} \neq B_{\ell-1} \cdot h_\ell / t_\ell$,

(b) $A_{\ell-1} = B_{\ell-1} \cdot h^*/t^*$ and $h_\ell / t_\ell = h^*/t^*$, or

(c) $B_\ell = 0$.

By repeating the above argument, and because we are assuming that (3) does not hold, we see that either

(i) $A_{i-1} \neq B_{i-1} \cdot h_i / t_i$ and $A_i = B_i \cdot h^*/t^*$ for some $i = 1, \ldots, \ell$, or

(ii) $B_i = 0$ for some $i = 1, \ldots, \ell$.

If we wish, we can categorize these as Type IIc(i) and IIc(ii) forgeries. Note that for a Type IIc(i) forgery, we may also assume that $h_j / t_j = h^*/t^*$ for $j = i+1, \ldots, \ell$, but we do not use this fact here.

The probability if a Type IIc(i) forgery can be bounded by $UN \cdot \text{RPR}\mathsf{adv}[\mathcal{B}_{\text{IIc}}, \mathit{Hash}] + O(UN/q)$. The random-preimage adversary $\mathcal{B}_{\text{IIc}}$ works by **guessing $\mathcal{R}^*$ and then guessing the index i at which condition (i) above occurs**. Analogous to (6), we have

$$(A_{i-1} - B_{i-1} h_i / t_i) = (B_{i-1} + s_i^{-1} c_i t_i)(h^*/t^* - h_i / t_i). \tag{7}$$

At the time the substitution $\mathtt{R}_i \mapsto s_i^{-1}(h_i + t_i \mathtt{D})$ is made, all of the terms appearing in (7), besides s_i and h^*, are already fixed. Moreover, we are assuming the left hand side of (7) is nonzero. This implies there is a one-to-one correspondence: for every h^* such that $h^*/t^* - h_i / t_i \neq 0$ there exists a unique s_i^{-1} such that $B_{i-1} + s_i^{-1} c_i t_i \neq 0$ and *vice versa*. Adversary $\mathcal{B}_{\text{IIc}}$ uses its challenge as the value of h^* and solves (7) for s_i^{-1}. Note that there are (at most) two values of h^* for which this will fail, one that satisfies $h^*/t^* - h_i / t_i = 0$ and the other that makes $s_i^{-1} = 0$.

The probability of a Type IIc(ii) forgery is easily seen to be at most $(UN)/(q-1)$. □

NOTES:

1. This scheme cannot be secure if we allow raw signing queries. Here is one simple attack. Suppose we get a presignature $\mathcal{R}$ with $t \coloneqq \bar{C}(\mathcal{R})$ and we compute $\mathcal{R}^* = 2\mathcal{R}$. Let $h^* = \mathit{Hash}(m^*)$ be the hash of a message m^* for which we want to forge a signature. We solve $h/t = h^*/t^*$ for h and ask for a raw signature on h using presignature $\mathcal{R}$, obtaining the signature (s, t). We then compute s^* satisfying $(s^*)^{-1}t^* = cts^{-1}$, so (s^*, t^*) is a forgery on m^*.

2. More generally, we really do need to assume that given t and t^*, it is hard to find preimages of h and h^* such that $h/t = h^*/t^*$ holds, as otherwise, essentially the same attack can be applied. Thus, ratio resistance is essential.

3. An attacker could try the above attack with $\mathcal{R}^* = 2\mathcal{R}, 3\mathcal{R}, \ldots$, obtaining many candidates for t^* to combine with many candidates for h and h^*. This would give us a multiplicative version of the 3-sum problem, for which there is no known attack that is significantly better than birthday (see [NS15]).

4. It is easy to see that the result of Theorem 3 applies equally well to *strong unforgeability up to sign* (see Section 4.1.1). The reasoning is the same is in the discussion following Theorem 1.

7.4 ECDSA with presignatures and additive key derivation

Now suppose we combine presignatures with additive key derivation. Here, we assume that `presig` queries take no input as before, but the signing queries take as input an index k that specifies the presignature to use, along with a message m_k and the tweak e_k.

We define $\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{akd,ps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}, \mathfrak{E}]$ to be adversary $\mathcal{A}$'s advantage in winning this modified CMA game in the EC-GGM. We can still prove security of ECDSA in this setting using stronger intractability assumptions for *Hash*.

Let us first consider the symbolic simulation of the signing oracle. Using the notation established above, $h_k \coloneqq \mathit{Hash}(m_k)$ and $t_k \coloneqq \bar{C}(\mathcal{R}_k)$. We want to choose $s_k \in \mathbb{Z}_q^*$ at random and then substitute $s_k^{-1}(h_k + t_k e_k + t_k \mathtt{D})$, rather than $s_k^{-1}(h_k + t_k \mathtt{D})$ for $\mathtt{R}_k$ in all polynomials in $\mathit{Domain}(\pi)$ that involve $\mathtt{R}_k$. The proof of Lemma 2 goes through unchanged.

Definition 8 (4sum1 intractability). *Let Hash be a hash function whose output space is $\mathbb{Z}_q$. Let $\mathfrak{E} \subseteq \mathbb{Z}_q$. Let $\mathcal{A}$ be an adversary. We define* $\text{4sum1}\mathsf{adv}[\mathcal{A}, \mathit{Hash}, \mathfrak{E}]$ *to be the advantage of $\mathcal{A}$ in breaking the* ***4sum1 property*** *of Hash with respect to the set $\mathfrak{E}$. This is defined as the probability that $\mathcal{A}$ wins the following game.*

- *The challenger chooses $t \in \mathbb{Z}_q$ uniformly at random and gives t to $\mathcal{A}$.*
- *$\mathcal{A}$ outputs m, e, m^*, e^*, where $e, e^* \in \mathfrak{E}$.*
- *We say $\mathcal{A}$* ***wins the game*** *if*

$$h + te = h^* + te^*,$$

where $e \neq e^*$ and $h \coloneqq \mathit{Hash}(m)$ and $h^* \coloneqq \mathit{Hash}(m^*)$.

Definition 9 (4sum2 intractability). *Let Hash be a hash function whose output space is $\mathbb{Z}_q$. Let $\mathfrak{E} \subseteq \mathbb{Z}_q$. Let $\mathcal{A}$ be an adversary. We define* $4\text{sum}2\mathsf{adv}[\mathcal{A}, \mathit{Hash}, \mathfrak{E}]$ *to be the advantage of $\mathcal{A}$ in breaking the **4sum2 property** of Hash with respect to the set $\mathfrak{E}$. This is defined as the probability that $\mathcal{A}$ wins the following game.*

- *The adversary asks the challenger for many random samples in $\mathbb{Z}_q^*$, and the adversary chooses one such sample $t \in \mathbb{Z}_q^*$.*
- *The challenger chooses $t^* \in \mathbb{Z}_q^*$ at random and gives t^* to $\mathcal{A}$.*
- *$\mathcal{A}$ outputs m, e, m^*, e^*, where $e, e^* \in \mathfrak{E}$.*
- *We say $\mathcal{A}$ **wins the game** if*
$$h/t + e = h^*/t^* + e^*,$$
where $h := \mathit{Hash}(m)$ and $h^* := \mathit{Hash}(m^*) \neq 0$.

Theorem 4. *Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\text{ecdsa}}$ as in Definition 2 with **additive key derivation and presignatures** that makes at most N presignature, signing, or group queries, of which N_{psig} are presignature requests. Let U denote the maximum number of* unused *presignatures at any point in time. Then there exist adversaries $\mathcal{B}_{\text{Ia}}$, $\mathcal{B}_{\text{Ib}}$, $\mathcal{B}_{\text{IIa}}$, $\mathcal{B}_{\text{IIb}}$, and $\mathcal{B}_{\text{IIc}}$, and $\mathcal{B}_{\text{III}}$, whose running times are essentially the same as $\mathcal{A}$ plus $O(UN)$, such that*

$$\begin{aligned}\text{CMA}^{\text{ggm}}_{\text{akd,ps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{ecdsa}}, \mathfrak{E}] \leq\ & \text{CR}\mathsf{adv}[\mathcal{B}_{\text{Ia}}, \mathit{Hash}] + \\ & (4 + o(1))N_{\text{psig}} \cdot 4\text{sum}1\mathsf{adv}[\mathcal{B}_{\text{Ib}}, \mathit{Hash}, \mathfrak{E}] + \\ & (4 + o(1))N|\mathfrak{E}| \cdot \text{RPR}\mathsf{adv}[\mathcal{B}_{\text{IIa}}, \mathit{Hash}] + \\ & (4 + o(1))N \cdot 4\text{sum}2\mathsf{adv}[\mathcal{B}_{\text{IIb}}, \mathit{Hash}, \mathfrak{E}] + \\ & UN|\mathfrak{E}| \cdot \text{RPR}\mathsf{adv}[\mathcal{B}_{\text{IIc}}, \mathit{Hash}] + \\ & \text{ZPR}\mathsf{adv}[\mathcal{B}_{\text{III}}, \mathit{Hash}] + \\ & O(N^2/q).\end{aligned}$$

Also, adversary $\mathcal{B}_{\text{IIb}}$ obtains $O(N_{\text{psig}})$ random samples from its challenger.

Proof. We categorize forgeries as Types Ia, Ib, IIa, IIb, IIc, and III: Types Ia and Ib are as in Theorem 2, Types IIa–IIc are as in Theorem 3, and Type III is as in Theorem 1.

The analysis we did for Type Ia and III forgeries in Section 6 goes through here without any change. Also, the analysis we did for Type II forgeries in Section 6 carries over here for Type IIa forgeries.

Type Ib forgeries. We get a Type Ib forgery if and only if the equation (1) holds with $e \neq e^*$. Without presignatures, the adversary had to commit to h and e before learning t, but with presignatures, the adversary is free to choose h and e, along with h^* and e^*, *after* learning t. Indeed, we see that creating a Type Ib forgery is essentially equivalent to breaking the 4sum1 property in Definition 8. We can easily use such a forger to break the 4sum1 property as follows: given the challenge t in the 4sum1 game, we **guess the relevant presignature**, set $t_k := t$ and run the sampler on t to get $\mathcal{R}_k$. This gives us $\mathcal{B}_{\text{Ib}}$ in Theorem 4.

Type IIb and IIc forgeries. Everything goes through exactly as in Theorem 3, but with h_i replaced by $\Delta_i := h_i + t_i e_i$ and h^* replaced by $\Delta^* := h^* + t^* e^*$. In particular, we categorize Type IIb forgeries as those where

$$\frac{\Delta_1}{t_1} = \cdots = \frac{\Delta_k}{t_k} = \frac{\Delta^*}{t^*}.$$

We can easily use a Type IIb forger to break the 4sum2 property as follows. In the first stage of the attack game in Definition 9, we use the random samples given by the 4sum2-challenger to generate all the presignatures we need using the sampling algorithm. With overwhelming probability, $O(N_{\text{psig}})$ random samples will suffice. We then **guess the group operation that produces $\mathcal{R}^*$**. At the time this group group operation is performed, we choose one of the variables $\mathtt{R}_i$ appearing in $\pi^{-1}(\mathcal{R}^*)$ arbitrarily and select t in the attack game in Definition 9 to the corresponding sample t_i. We then obtain t^* from our 4sum2-challenger and run the sampling algorithm on t^* to get $\mathcal{R}^*$. A Type IIb forgery will give us the values m, e, m^*, e^* we need to win the attack game in Definition 9. This is adversary $\mathcal{B}_{\text{IIb}}$ in Theorem 4.

The adversary $\mathcal{B}_{\text{IIc}}$ in Theorem 4 is exactly the same as $\mathcal{B}_{\text{IIc}}$ in Theorem 3, but with h_k replaced by Δ_k and h^* replaced by Δ^*, and where we also have to guess the tweak e^*. □

NOTES:

1. Just as in the case of presignatures without additive key derivation, this scheme cannot be secure if we allow raw signing queries.

2. It is easy to see that the result of Theorem 4 applies equally well to *strong unforgeability up to sign* (see Section 4.1.1). The reasoning is the same is in the discussion following Theorem 2.

7.4.1 How strong are the 4sum1 and 4sum2 properties?

Consider first the 4sum1 property. If we just choose e and e^* arbitrarily, then viewing *Hash* as a random oracle, then analogous to the birthday attack, we can find m and m^* satisfying the required relation in time $O(\sqrt{q})$. However, by exploiting the fact that we also have control over e and e^*, we can beat the birthday attack.

Indeed, suppose we view *Hash* as a random oracle, and the elements of $\mathfrak{E}$ are randomly chosen. Then this problem is no harder than the 4-sum problem studied in Wagner [Wag02] and elsewhere [BLN+09, NS15]. Wagner gave an algorithm to solve this problem that beats the birthday attack. In Appendix C, we sketch Wagner's algorithm, adapted to our setting. One consequence of this is that if $|\mathfrak{E}| = \Theta(q^{1/3})$, then we can solve this 4-sum problem and forge a signature in time $O(q^{1/3})$. The attack works as follows.

- Make one presignature query to get the group element $\mathcal{R}$ and let $t := \bar{C}(\mathcal{R})$.

- Use Wagner's algorithm to find m, e, m^*, e^* such that $h + te = h^* + te^*$, where $e \neq e^*$ and $h := \mathit{Hash}(m)$ and $h^* := \mathit{Hash}(m^*)$.

- Now ask for a signature using this presignature on message m with tweak e.

- This signature is also a signature on m^* with tweak e^*.

The $O(q^{1/3})$ work is time spent computing hashes of messages and tweaks (which themselves may well just be hashes), and performing hash table lookups. Mitigating against this attack is

- the fact that the $O(q^{1/3})$ time must be done *between* the time that the presignature is generated and the time that the adversary asks for a signature using that presignature, and
- the fact that the attack takes space $O(q^{1/3})$ — but see [BLN+09, NS15] for time-space trade-offs.

We stress that this $O(q^{1/3})$ attack requires just one presignature and one corresponding signature.

It is also easily seen that the 4sum2 property is also no harder than a 4-sum problem.

7.4.2 Alternative analysis

Just as we did in Section 6.1, we can consider the setting where tweaks are derived from identifiers via a hash function *Hash′*, without any further restrictions.

In this setting, the term

$$\mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{Ia}}, \mathit{Hash}]$$

appearing in Theorem 4 would have to be replaced by

$$\mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{Ia}}, \mathit{Hash}] + \mathrm{CR}\mathsf{adv}[\mathcal{B}'_{\mathrm{Ia}}, \mathit{Hash}'].$$

Also in this setting, the term

$$|\mathfrak{E}| \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIa}}, \mathit{Hash}]$$

appearing in Theorem 4 can be replaced by

$$\mathrm{2sum}\mathsf{adv}[\mathcal{B}_{\mathrm{IIa}}, \mathit{Hash}, \mathit{Hash}'].$$

As for the term

$$|\mathfrak{E}| \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIc}}, \mathit{Hash}]$$

appearing in Theorem 4, with a bit more work, one can show that this can in fact be replaced by

$$(4 + o(1)) \cdot \mathrm{2sum}\mathsf{adv}[\mathcal{B}_{\mathrm{IIc}}, \mathit{Hash}, \mathit{Hash}'].$$

Here, the value e in the definition will be set to a random value, which represents the value

$$(A_{i-1} - B_{i-1}\Delta_i/t_i)/(B_{i-1} + s_i^{-1}c_i t_i) + \Delta_i/t_i$$

in the proof; h in the definition is set to 0; t in the definition would correspond to t^* in the proof.

Also, for the terms in Theorem 4 involving 4sum1 and 4sum2 intractability, one have to replace the corresponding security definitions so that instead of giving $e, e^* \in \mathfrak{E}$, the adversary would have to give preimages of e and e^*.

8 ECDSA with re-randomized presignatures

We saw the ECDSA with presignatures leads to potential vulnerabilities, especially when combined with additive key derivation. At the very least, we require additional intractability assumptions. In this section, we explore a variant in which the presignatures are **re-randomized** when used for signing. For threshold ECDSA implementations, this re-randomization maintains most of the benefits of presignatures; however, it also maintains most of the security properties that we had without presignatures, both in the settings with and without additive key derivation.

So now a presignature is of the form

$$(r', \mathcal{R}'),$$

where $r' \xleftarrow{\$} \mathbb{Z}_q$ and $\mathcal{R}' \leftarrow r'\mathcal{G}$. As before, when processing a request to sign a message, we can allocate one such precomputed pair and use it to finish the computation of the signature. However, instead of using the presignature directly, we *re-randomize* it, computing $\delta \xleftarrow{\$} \mathbb{Z}_q$, and using

$$(r, \mathcal{R}) := (r' + \delta, \mathcal{R}' + \delta\mathcal{G})$$

as the presignature. Crucially, the value of δ is given to the adversary as an output of the signing request.

NOTES:

1. The reason why we insist on giving δ to the adversary is that a protocol implementing a distributed signing service may ultimately reveal δ. This allows us to reduce the security of such a distributed protocol to this primitive. Depending on how the distributed signing service is implemented, generating δ may or may not introduce extra latency.

2. Instead of generating δ at random, it could also be obtained by deriving it as a hash of $\mathcal{R}'$ and the signing request. The results we present here could be adapted to this setting, especially if we model the hash as a random oracle. While the security results would be somewhat weaker than if δ is generated at random, they would still be significantly stronger than not using any re-randomization at all.

8.1 A lazy simulation of the signature attack game

We start with the analog of Lazy-Sim in Section 7.1, but now with re-randomized presignatures. Figure 6 gives the details of **Lazy-Sim**.

8.2 A symbolic simulation of the signature attack game

We define a **symbolic** simulation of the attack game, which is the analog of Symbolic-Sim in Section 7.2. As in Section 7.2, *Domain*(π) will now consist of polynomials of the form

$$a + b\mathtt{D} + c_1\mathtt{R}_1 + c_2\mathtt{R}_2 + \cdots ,$$

```
1. Initialization:
    (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
    (b) $d \xleftarrow{\$} \mathbb{Z}_q^*$
    (c) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
    (d) invoke $(\texttt{map}, d)$ to obtain $\mathcal{D}$
    (e) $k \leftarrow 0$; $K \leftarrow \emptyset$
    (f) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
    (a) if $i \notin \mathit{Domain}(\pi)$:
        i. $\mathcal{P} \xleftarrow{\$} E^*$;
           while $\mathcal{P} \in \mathit{Range}(\pi)$ do: $\mathcal{P} \xleftarrow{\$} E^*$
        ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
    (b) return $\pi(i)$
3. To process a group oracle query $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2)$:
    (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
        i. $i \xleftarrow{\$} \mathbb{Z}_q^*$;
           while $i \in \mathit{Domain}(\pi)$ do: $i \xleftarrow{\$} \mathbb{Z}_q^*$
        ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
    (b) invoke $(\texttt{map}, \pi^{-1}(\mathcal{P}_1) + \pi^{-1}(\mathcal{P}_2))$ and return the result
4. To process a presignature request:
    (a) $k \leftarrow k + 1$
    (b) $r'_k \xleftarrow{\$} \mathbb{Z}_q$
    (c) invoke $(\texttt{map}, r'_k)$ to get $\mathcal{R}'_k$
    (d) $K \leftarrow K \cup \{k\}$;  return $\mathcal{R}'_k$
5. To process a request to sign $m_k$ using presignature number $k \in K$:
    (a) $K \leftarrow K \setminus \{k\}$
    (b) $\delta_k \xleftarrow{\$} \mathbb{Z}_q$
    (c) $r_k \leftarrow r'_k + \delta_k$
    (d) if $r_k = 0$ then return *fail*
    (e) invoke $(\texttt{map}, r_k)$ to get $\mathcal{R}_k$
    (f) $t_k \leftarrow \bar{C}(\mathcal{R}_k) \in \mathbb{Z}_q$
    (g) $h_k \leftarrow \mathit{Hash}(m_k) \in \mathbb{Z}_q$
    (h) if $t_k = 0$ or $h_k + t_k d = 0$ then return *fail*
    (i) $s_k \leftarrow r_k^{-1}(h_k + t_k d)$
    (j) return $(s_k, t_k, \mathcal{R}_k, \delta_k)$
```

Figure 6: Lazy-Sim (with re-randomized presignatures)

where $a, b, c_1, c_2, \ldots \in \mathbb{Z}_q$, and $\mathtt{D}, \mathtt{R}_1, \mathtt{R}_2, \ldots$ are indeterminants. Here, $\mathtt{D}$ symbolically represents the value of d, and $\mathtt{R}_k$ symbolically represents the value of r'_k (and *not* r_k). Figure 7 gives the details of **Symbolic-Sym**.

Lemma 3. *The difference between the adversary's forging advantage in the Lazy-Sim and Symbolic-Sim games (as described in Sections 8.1 and 8.2) is $O(N^2/q)$.*

The proof of Lemma 3 follows the same lines as that of Lemma 2, and we leave the details to the reader.

8.3 Security of ECDSA with re-randomized presignatures

We define $\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{rrps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}, \mathfrak{E}]$ to be adversary $\mathcal{A}$'s advantage in winning this modified CMA game in the EC-GGM.

Theorem 5. *Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\mathrm{ecdsa}}$ as in Definition 2 with* ***re-randomized presignatures*** *that makes at most N presignature, signing, or group queries. Let U denote the maximum number of* unused *presignatures at any point in time. Then there exist adversaries $\mathcal{B}_{\mathrm{I}}$, $\mathcal{B}_{\mathrm{IIa}}$, $\mathcal{B}_{\mathrm{IIbc}}$, and $\mathcal{B}_{\mathrm{III}}$, whose running times are essentially the same as $\mathcal{A}$ plus*

```
1. Initialization:
    (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
    (b) invoke $(\mathtt{map}, 1)$ to obtain $\mathcal{G}$
    (c) invoke $(\mathtt{map}, \mathtt{D})$ to obtain $\mathcal{D}$
    (d) $k \leftarrow 0$; $K \leftarrow \emptyset$
    (e) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\mathtt{map}, i)$:
    (a) if $i \notin \mathit{Domain}(\pi)$:
        i. $\mathcal{P} \xleftarrow{\$} E^*$;
           if $\mathcal{P} \in \mathit{Range}(\pi)$ then abort
        ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
    (b) return $\pi(i)$
3. To process a group oracle query $(\mathtt{add}, \mathcal{P}_1, \mathcal{P}_2)$:
    (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
        i. $i \xleftarrow{\$} \mathbb{Z}_q^*$;
           if $i \in \mathit{Domain}(\pi)$ then abort
        ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
    (b) invoke $(\mathtt{map}, \pi^{-1}(\mathcal{P}_1) + \pi^{-1}(\mathcal{P}_2))$ and return the result
4. To process a presignature request:
    (a) $k \leftarrow k + 1$
    (b) invoke $(\mathtt{map}, \mathtt{R}_k)$ to get $\mathcal{R}'_k$
    (c) $K \leftarrow K \cup \{k\}$; return $\mathcal{R}'_k$
5. To process a request to sign $m_k$ using presignature number $k \in K$:
    (a) $K \leftarrow K \setminus \{k\}$
    (b) $\delta_k \xleftarrow{\$} \mathbb{Z}_q$
    (c) if $\mathtt{R}_k + \delta_k \in \mathit{Domain}(\pi)$ then abort
    (d) invoke $(\mathtt{map}, \mathtt{R}_k + \delta_k)$ to obtain $\mathcal{R}_k$
    (e) $t_k \leftarrow \bar{C}(\mathcal{R}_k)$
    (f) if $t_k = 0$ then abort
    (g) $h_k \leftarrow \mathit{Hash}(m_k) \in \mathbb{Z}_q$
    (h) $s_k \xleftarrow{\$} \mathbb{Z}_q^*$
    (i) substitute $s_k^{-1}(h_k + t_k\mathtt{D}) - \delta_k$ for $\mathtt{R}_k$ throughout $\mathit{Domain}(\pi)$, and abort if any two polynomials collapse
    (j) return $(s_k, t_k, \mathcal{R}_k, \delta_k)$
```

Figure 7: Symbolic-Sim (with re-randomized presignatures)

$O(UN)$, such that

$$
\begin{aligned}
\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{rrps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{ecdsa}}] \le {} & \mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{I}}, \mathit{Hash}] + \\
& (4 + o(1))N \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIa}}, \mathit{Hash}] + \\
& N \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIbc}}, \mathit{Hash}] + \\
& \mathrm{ZPR}\mathsf{adv}[\mathcal{B}_{\mathrm{III}}, \mathit{Hash}] + \\
& O(N^2/q).
\end{aligned}
$$

Proof. We categorize forgeries just as in Theorem 3, but we lump Types IIb and IIc into a single Type IIbc. Forgeries of types I, IIa, and III are handled just as in Theorem 3.

For forgeries of type IIbc, just as in Theorem 3, we suppose that at the time $\mathcal{R}^*$ is initially generated, we have

$$\pi^{-1}(\mathcal{R}^*) = a + b\mathtt{D} + c_1\mathtt{R}_1 + \cdots + c_\ell\mathtt{R}_\ell,$$

where the c_i's are nonzero; however, during the attack, we substitute

$$\mathtt{R}_i \mapsto s_i^{-1}(h_i + t_i\mathtt{D}) - \delta_i \quad \text{for } i = 1, \ldots, \ell,$$

again, in that order. For $i = 0, \ldots, \ell$, define

$$A_i := a + \sum_{j \le i} c_j(h_j/s_j - \delta_j) \quad \text{and} \quad B_i := b + \sum_{j \le i} c_j t_j/s_j.$$

Equation (6) then becomes

$$(A_{\ell-1} - B_{\ell-1}h_\ell/t_\ell - c_\ell\delta_\ell) = (B_{\ell-1} + s_\ell^{-1}c_\ell t_\ell)(h^*/t^* - h_\ell/t_\ell). \tag{8}$$

At the time the substitution $\mathtt{R}_\ell \mapsto s_\ell^{-1}(h_\ell + t_\ell \mathtt{D}) - \delta_\ell$ is made, all of the terms appearing in (8), besides δ_ℓ, s_ℓ, and h^*, are already fixed. Therefore, the left-hand side of (8) will vanish with probability $1/q$, and as long as this does not happen, we can use this Type IIbc forger to break random-preimage resistance. Indeed, just as we argued in the proof of Theorem 3, there is a one-to-one correspondence: for every h^* such that $h^*/t^* - h_\ell/t_\ell \neq 0$ there exists a unique s_ℓ^{-1} such that $B_{\ell-1} + s_\ell^{-1} c_\ell t_\ell \neq 0$ and *vice versa*. We use this the given random-preimage challenge as the value of h^* and solve (8) for s_ℓ^{-1}. □

NOTES:

1. With re-randomized presignatures, we again obtain security with respect to raw signing queries (allowing arbitrary, unconstrained $h_k \in \mathbb{Z}_q$).

2. One sees from the proof of Theorem 5 that we only need that the randomizer δ_k is sufficiently unpredictable — it need not be uniformly distributed over $\mathbb{Z}_q$.

3. It is easy to see that the result of Theorem 5 applies equally well to *strong unforgeability up to sign* (see Section 4.1.1).

8.4 ECDSA with re-randomized presignatures and additive key derivation

Now suppose we combine re-randomized presignatures with additive key derivation. We define $\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{akd,rrps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}, \mathfrak{C}]$ to be adversary $\mathcal{A}$'s advantage in winning this modified CMA game in the EC-GGM.

Theorem 6. *Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\mathrm{ecdsa}}$ as in Definition 2 with* ***additive key derivation and re-randomized presignatures*** *that makes at most N presignature, signing, or group queries, of which N_{psig} are presignature queries. Let U denote the maximum number of* unused *presignatures at any point in time. Then there exist adversaries $\mathcal{B}_{\mathrm{Ia}}$, $\mathcal{B}_{\mathrm{Ib}}$, $\mathcal{B}_{\mathrm{IIa}}$, $\mathcal{B}_{\mathrm{IIc}}$, and $\mathcal{B}_{\mathrm{III}}$, whose running times are essentially the same as $\mathcal{A}$ plus $O(UN)$, such that*

$$\begin{aligned}\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{akd,rrps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{ecdsa}}, \mathfrak{C}] \leq {} & \mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{Ia}}, \mathit{Hash}] + \\ & (4 + o(1))N_{\mathrm{sig}}|\mathfrak{C}| \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{Ib}}, \mathit{Hash}] + \\ & (4 + o(1))N|\mathfrak{C}| \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIa}}, \mathit{Hash}] + \\ & N|\mathfrak{C}| \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIbc}}, \mathit{Hash}] + \\ & \mathrm{ZPR}\mathsf{adv}[\mathcal{B}_{\mathrm{III}}, \mathit{Hash}] + \\ & O(N^2/q).\end{aligned}$$

Proof. Forgeries are categorized just as in Theorem 4, but we lump Types IIb and IIc into a single Type IIbc. Type Ia and Ib forgeries are handled just as in Theorem 2. Type IIa forgeries are handled just like Type II forgeries in Theorem 2. Type III forgeries are handled just as in Theorem 1.

For Type IIbc forgeries, everything goes through exactly as in Theorem 5, but with h_i replaced by $\Delta_i := h_i + t_i e_i$ and h^* replaced by $\Delta^* := h^* + t^* e^*$, and the adversary $\mathcal{B}_{\mathrm{IIbc}}$ has to guess e^*. □

NOTES:

1. With re-randomized presignatures, we again obtain security with respect to raw signing queries (allowing arbitrary, unconstrained $h_k, e_k \in \mathbb{Z}_q$).

2. Just in in Theorem 5, it is not essential that δ_k is uniformly distributed over $\mathbb{Z}_q$ — it only needs to be sufficiently unpredictable.

3. It is easy to see that the result of Theorem 6 applies equally well to *strong unforgeability up to sign* (see Section 4.1.1).

8.4.1 Alternative analysis

Just as we did in Sections 6.1 and 7.4.2, we can consider the setting where tweaks are derived from identifiers via a hash function Hash', without any further restrictions.

The terms in Theorem 6 involving $\mathcal{B}_{\text{Ia}}$ and $\mathcal{B}_{\text{Ib}}$ are replaced as in Section 6.1. The term involving $\mathcal{B}_{\text{IIa}}$ is replaced as in Section 7.4.2. The term involving $\mathcal{B}_{\text{IIbc}}$ is replaced in the same way the term involving $\mathcal{B}_{\text{IIc}}$ is replaced in Section 7.4.2.

9 Homogeneous key derivation

We propose a new key derivation technique (a similar construction was given in [GS14] for completely different purposes). This derivation technique is still essentially linear, and so enjoys many of the same advantages of additive key derivation, including

- the ability to derive public keys from a master public key, and
- the ability to efficiently implement the scheme as a threshold signature scheme.

The basic idea is this. The master secret key is now a random pair $(d, d') \in \mathbb{Z}_q \times \mathbb{Z}_q$, and the corresponding master public key is the pair

$$(\mathcal{D}, \mathcal{D}') := (d\mathcal{G}, d'\mathcal{G}) \in E \times E.$$

For a given "tweak" $e \in \mathbb{Z}_q$, the corresponding derived secret key is $d + ed' \in \mathbb{Z}_q$ and the corresponding derived public key is $\mathcal{D} + e\mathcal{D}'$.

We consider homogeneous key derivation without presignatures, with presignatures, and with re-randomized presignatures.

As we will see, we can prove stronger results with homogeneous key derivation than we could with additive key derivation. In particular, we will not need to assume that the tweaks come from some predetermined set $\mathfrak{E} \subseteq \mathbb{Z}_q$. As such, we will assume that a tweak $e \in \mathbb{Z}_q$ is derived from the hash function *Hash* as

$$e \leftarrow \mathit{Hash}(id),$$

where id is an arbitrary identifier. Here, *Hash* is the same hash function used by ECDSA; however, it could also be a different hash function (the only requirement is that this hash function maps into $\mathbb{Z}_q$ and is collision resistant). The signing algorithm will take as input both a message m and an identifier id. In the forgery attack game, a forgery consists of a valid signature (s^*, t^*) on a message m^* and an identifier id^*, subject to the constraint that the signing oracle was not invoked with the same message/identifier pair (m^*, id^*).

9.1 Homogeneous key derivation without presignatures

The lazy simulation in Section 5.1 is modified as follows:

- In the initialization step, the challenger chooses $(d, d') \in \mathbb{Z}_q \times \mathbb{Z}_q$ at random, invokes $(\texttt{map}, d)$ and $(\texttt{map}, d')$ to obtain $\mathcal{D}$ and $\mathcal{D}'$. The challenger gives $(\mathcal{G}, \mathcal{D}, \mathcal{D}')$ to the adversary.

- In a signing request, the adversary supplies an identifier id in addition to a message m, and the tweak $e \in \mathbb{Z}_q$ is computed as $e \leftarrow \mathit{Hash}(id)$. To process such a signing request, the challenger carries out the same logic, but with $d + ed'$ replacing d in steps 4(f) and 4(g).

To verify a signature with respect to a tweak e^*, where $e^* := \mathit{Hash}(id^*)$, the signature is verified with respect to the public key $\mathcal{D} + e^*\mathcal{D}'$.

The symbolic simulation in Section 5.2 is modified as follows:

- In the initialization step, the challenger invokes $(\texttt{map}, \texttt{D})$ and $(\texttt{map}, \texttt{D}')$ to obtain $\mathcal{D}$ and $\mathcal{D}'$. The challenger gives $(\mathcal{G}, \mathcal{D}, \mathcal{D}')$ to the adversary. Here, $\texttt{D}$ and $\texttt{D}'$ are distinct indeterminants.

- In a signing request, the adversary supplies an identifier id in addition to a message m, and the tweak $e \in \mathbb{Z}_q$ is computed as $e \leftarrow \mathit{Hash}(id)$. To process such a signing request, the challenger carries out the same logic, but with $\texttt{D} + e\texttt{D}'$ replacing $\texttt{D}$ in step 4(g).

We define $\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{hkd}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ to be adversary $\mathcal{A}$'s advantage in winning this modified CMA game in the EC-GGM.

It is easy to prove that Lemma 1 carries over to this setting without change. We leave this to the reader.

We can prove the following analog of Theorem 2. As the reader will notice, the statement of this theorem is almost the same as Theorem 1.

Theorem 7. *Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\mathrm{ecdsa}}$ as in Definition 2 with* ***homogeneous key derivation*** *that makes at most N signing or group queries. Then there exist adversaries $\mathcal{B}_{\mathrm{I}}$, $\mathcal{B}_{\mathrm{II}}$, and $\mathcal{B}_{\mathrm{III}}$, whose running times are essentially the same as $\mathcal{A}$, such that*

$$\begin{aligned}\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{hkd}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{ecdsa}}] \leq\ & \mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{I}}, \mathit{Hash}] + \\ & (4 + o(1))N \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{II}}, \mathit{Hash}] + \\ & \mathrm{ZPR}\mathsf{adv}[\mathcal{B}_{\mathrm{III}}, \mathit{Hash}] + \\ & O(N^2/q).\end{aligned}$$

Proof. We categorize forgeries as Type I, II, or III just as in Theorem 1.

For a Type I forgery, for $\eta \in \{\pm 1\}$, we have

$$(s^*)^{-1}(h^* + t\texttt{D} + te^*\texttt{D}') = \eta s^{-1}(h + t\texttt{D} + te\texttt{D}').$$

This gives us three equations:

$$(s^*)^{-1}h^* = \eta s^{-1}h, \quad (s^*)^{-1}t = \eta s^{-1}t, \quad \text{and} \quad (s^*)^{-1}te^* = \eta s^{-1}te.$$

These three equations imply

$$h^* = h \quad \text{and} \quad e^* = e.$$

This immediately gives us the adversary $\mathcal{B}_{\text{I}}$ in Theorem 7 that breaks the collision resistance of *Hash*, either of the form $\mathit{Hash}(m^*) = \mathit{Hash}(m)$ or $\mathit{Hash}(id^*) = \mathit{Hash}(id)$.

For a Type II forgery, if $\pi^{-1}(\mathcal{R}^*) = a + b\mathtt{D} + b'\mathtt{D}'$, we have

$$a + b\mathtt{D} + b'\mathtt{D}' = (s^*)^{-1}(h^* + t^*\mathtt{D} + t^*e^*\mathtt{D}').$$

This gives us three equations:

$$a = (s^*)^{-1}h^*, \quad b = (s^*)^{-1}t^*, \quad \text{and} \quad b' = (s^*)^{-1}t^*e^*.$$

Just as in Theorem 1, we obtain $b \neq 0$, $a \neq 0$, and

$$t^* = h^*a^{-1}b.$$

In addition, we have $b' = be^*$. So just as in Theorem 1, we obtain an adversary $\mathcal{B}_{\text{II}}$ that breaks the random-preimage resistance of *Hash*.

For a Type III forgery, just as in Theorem 1, we obtain an adversary $\mathcal{B}_{\text{III}}$ that breaks the zero-preimage resistance of *Hash*. □

NOTES:

1. The above analysis shows that the scheme is secure even with a "raw" signing oracle.

2. It is easy to see that the result of Theorem 7 applies equally well to *strong unforgeability up to sign* (see Section 4.1.1).

9.2 Homogeneous key derivation with presignatures

The lazy simulation in Section 7.1 is modified as follows:

- In the initialization step, the challenger chooses $(d, d') \in \mathbb{Z}_q \times \mathbb{Z}_q$ at random, invokes $(\mathtt{map}, d)$ and $(\mathtt{map}, d')$ to obtain $\mathcal{D}$ and $\mathcal{D}'$. The challenger gives $(\mathcal{G}, \mathcal{D}, \mathcal{D}')$ to the adversary.

- In a signing request, the adversary supplies an identifier id_k in addition to a message m_k, and the tweak $e_k \in \mathbb{Z}_q$ is computed as $e_k \leftarrow \mathit{Hash}(id_k)$. To process such a signing request, the challenger carries out the same logic, but with $d + e_kd'$ replacing d in steps 5(c) and 5(d).

To verify a signature with respect to a tweak e^*, where $e^* := \mathit{Hash}(id^*)$, the signature is verified with respect to the public key $\mathcal{D} + e^*\mathcal{D}'$.

The symbolic simulation in Section 7.2 is modified as follows:

- In the initialization step, the challenger invokes $(\mathtt{map}, \mathtt{D})$ and $(\mathtt{map}, \mathtt{D}')$ to obtain $\mathcal{D}$ and $\mathcal{D}'$. The challenger gives $(\mathcal{G}, \mathcal{D}, \mathcal{D}')$ to the adversary. Here, $\mathtt{D}$ and $\mathtt{D}'$ are distinct indeterminants.

- In a signing request, the adversary supplies an identifier id_k in addition to a message m_k, and the tweak $e_k \in \mathbb{Z}_q$ is computed as $e_k \leftarrow \mathit{Hash}(id_k)$. To process such a signing request, the challenger carries out the same logic, but with $\mathtt{D} + e_k\mathtt{D}'$ replacing $\mathtt{D}$ in step 5(d).

We define $\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{hkd,ps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ to be adversary $\mathcal{A}$'s advantage in winning this modified CMA game in the EC-GGM.

It is easy to prove that Lemma 2 carries over to this setting without change. We leave this to the reader.

We can prove the following analog of Theorem 4. As the reader will notice, the statement of this theorem is almost the same as Theorem 3.

Theorem 8. *Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\mathrm{ecdsa}}$ as in Definition 2 with **homogenous key derivation and presignatures** that makes at most N presignature, signing, or group queries. Let U denote the maximum number of* unused *presignatures at any point in time. Then there exist adversaries $\mathcal{B}_{\mathrm{I}}$, $\mathcal{B}_{\mathrm{IIa}}$, $\mathcal{B}_{\mathrm{IIb}}$, $\mathcal{B}_{\mathrm{IIc}}$, and $\mathcal{B}_{\mathrm{III}}$, whose running times are essentially the same as $\mathcal{A}$ plus $O(UN)$, such that*

$$\begin{aligned}\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{hkd,ps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{ecdsa}}] \le \mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{I}}, \mathit{Hash}] + {}& \\ &(4 + o(1))N \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIa}}, \mathit{Hash}] + \\ &(4 + o(1))N \cdot \mathrm{RR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIb}}, \mathit{Hash}] + \\ &UN \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIc}}, \mathit{Hash}] + \\ &\mathrm{ZPR}\mathsf{adv}[\mathcal{B}_{\mathrm{III}}, \mathit{Hash}] + \\ &O(N^2/q).\end{aligned}$$

Proof. We categorize forgeries as Type I, IIa, IIb, IIc, or III essentially as in Theorem 3.

Everything goes through the same as in the proof of Theorem 7, except for the analysis of Type II forgeries.

Consider the point in time when the adversary queries the group oracle to obtain $\mathcal{R}^*$ for the first time. Let us call this a **Type IIa** forgery if at this time, $\pi^{-1}(\mathcal{R}^*)$ is of the form $a + b\mathtt{D} + b'\mathtt{D}'$. Type IIa forgeries can be dealt with in exactly the same way as Type II forgeries in the proof of Theorem 7.

Now, consider a Type II forgery that is not a Type IIa forgery. For such a forgery, the initial preimage of $\mathcal{R}^*$ is a polynomial that involves the indeterminants $\mathtt{R}_1, \mathtt{R}_2, \ldots$. However, before the attack ends, all of these variables must be substituted via signing queries, so that if the attack ends with a forgery, we must have $\pi^{-1}(\mathcal{R}^*) = (s^*)^{-1}(h^* + t^*\mathtt{D} + t^*e^*\mathtt{D}')$.

Just as in Theorem 3, we suppose that at the time $\mathcal{R}^*$ is initially generated, we have

$$\pi^{-1}(\mathcal{R}^*) = a + b\mathtt{D} + c_1\mathtt{R}_1 + \cdots + c_\ell\mathtt{R}_\ell,$$

where the c_i's are nonzero; however, during the attack, we substitute

$$\mathtt{R}_i \mapsto s_i^{-1}(h_i + t_i\mathtt{D} + t_ie_i\mathtt{D}') \quad \text{for } i = 1, \ldots, \ell,$$

again, in that order. For $i = 0, \ldots, \ell$, define

$$A_i := a + \sum_{j \le i} c_jh_j/s_j, \quad B_i := b + \sum_{j \le i} c_jt_j/s_j, \quad \text{and} \quad B'_i := b' + \sum_{j \le j} c_jt_je_j/s_j.$$

A forgery must satisfy:

$$A_\ell = (s^*)^{-1}h^*, \quad B_\ell = (s^*)^{-1}t^*, \quad \text{and} \quad B'_\ell = (s^*)^{-1}t^*e^*. \tag{9}$$

Note that the first of these two equations are identical to the two equations in (4) in the proof of Theorem 3. Indeed, we can complete the proof just as in Theorem 3, where Type IIb and IIc forgeries are defined in the same way.

□

NOTES:

1. Unlike as in Theorem 7, we see that this scheme is *insecure* if we allow a "raw" signing oracle.

2. It is easy to see that the result of Theorem 8 applies equally well to *strong unforgeability up to sign* (see Section 4.1.1).

9.3 Homogeneous key derivation with re-randomized presignatures

The lazy simulation in Section 8.1 is modified as follows:

- In the initialization step, the challenger chooses $(d, d') \in \mathbb{Z}_q \times \mathbb{Z}_q$ at random, invokes $(\texttt{map}, d)$ and $(\texttt{map}, d')$ to obtain $\mathcal{D}$ and $\mathcal{D}'$. The challenger gives $(\mathcal{G}, \mathcal{D}, \mathcal{D}')$ to the adversary.

- In a signing request, the adversary supplies an identifier id_k in addition to a message m_k, and the tweak $e_k \in \mathbb{Z}_q$ is computed as $e_k \leftarrow \mathit{Hash}(id_k)$. To process such a signing request, the challenger carries out the same logic, but with $d + e_k d'$ replacing d in steps 5(h) and 5(i).

To verify a signature with respect to a tweak e^*, where $e^* := \mathit{Hash}(id^*)$, the signature is verified with respect to the public key $\mathcal{D} + e^*\mathcal{D}'$.

The symbolic simulation in Section 8.2 is modified as follows:

- In the initialization step, the challenger invokes $(\texttt{map}, \texttt{D})$ and $(\texttt{map}, \texttt{D}')$ to obtain $\mathcal{D}$ and $\mathcal{D}'$. The challenger gives $(\mathcal{G}, \mathcal{D}, \mathcal{D}')$ to the adversary. Here, $\texttt{D}$ and $\texttt{D}'$ are distinct indeterminants.

- In a signing request, the adversary supplies an identifier id_k in addition to a message m_k, and the tweak $e_k \in \mathbb{Z}_q$ is computed as $e_k \leftarrow \mathit{Hash}(id_k)$. To process such a signing request, the challenger carries out the same logic, but with $\texttt{D} + e_k\texttt{D}'$ replacing $\texttt{D}$ in step 5(i).

We define $\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{hkd,rrps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ to be adversary $\mathcal{A}$'s advantage in winning this modified CMA game in the EC-GGM.

It is easy to prove that Lemma 3 carries over to this setting without change. We leave this to the reader.

We can prove the following analog of Theorem 6. As the reader will notice, the statement of this theorem is almost the same as Theorem 5.

Theorem 9. *Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\mathrm{ecdsa}}$ as in Definition 2 with* ***homogeneous key derivation and re-randomized presignatures*** *that makes at most N presignature, signing, or group queries. Let U denote the maximum number of* unused *presignatures at any point in time. Then there exist adversaries $\mathcal{B}_{\mathrm{I}}$, $\mathcal{B}_{\mathrm{IIa}}$, $\mathcal{B}_{\mathrm{IIc}}$, and $\mathcal{B}_{\mathrm{III}}$, whose running times are essentially the same as $\mathcal{A}$ plus $O(UN)$, such that*

$$\begin{aligned}\mathrm{CMA}^{\mathrm{ggm}}_{\mathrm{hkd,rrps}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{ecdsa}}] \le{} & \mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{I}}, \mathit{Hash}] + \\ & (4 + o(1))N \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIa}}, \mathit{Hash}] + \\ & N \cdot \mathrm{RPR}\mathsf{adv}[\mathcal{B}_{\mathrm{IIbc}}, \mathit{Hash}] + \\ & \mathrm{ZPR}\mathsf{adv}[\mathcal{B}_{\mathrm{III}}, \mathit{Hash}] + \\ & O(N^2/q).\end{aligned}$$

Proof. We categorize forgeries as Type I, IIa, IIbc, or III essentially as in Theorem 5.

The proof follows the same outline as that of Theorem 8, except for the analysis of Type IIbc forgeries, which follows the same outline as in Theorem 5. □

NOTES:

1. The above analysis shows that the scheme is secure even with a "raw" signing oracle.
2. It is easy to see that the result of Theorem 9 applies equally well to *strong unforgeability up to sign* (see Section 4.1.1).

Acknowledgments

Thanks to Yevgeniy Dodis for helpful discussions on public-use guarded random oracles. Thanks also to Nikolaos Makriyannis for bringing the paper [CMP20] to our attention, and for discussing their analysis of ECDSA with presignatures.

A Proof of Lemma 1

In order to make certain arguments simpler, we shall replace or lazy simulator by a slightly more lazy simulator. This simulator may `abort` under certain conditions, which means the entire experiment halts and no forgery is produced.

Lazy-Sim1:

```
1. Initialization:
    (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
    (b) $d \xleftarrow{\$} \mathbb{Z}_q^*$
    (c) invoke $(\mathtt{map}, 1)$ to obtain $\mathcal{G}$
    (d) invoke $(\mathtt{map}, d)$ to obtain $\mathcal{D}$
    (e) return $(\mathcal{G}, \mathcal{D})$
```

```
2. To process a group oracle query $(\texttt{map}, i)$:
   (a) if $i \notin \mathit{Domain}(\pi)$:
       i. $\mathcal{P} \xleftarrow{\$} E^*$;  if $\mathcal{P} \in \mathit{Range}(\pi)$ then abort
       ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
   (b) return $\pi(i)$

3. To process a group oracle query $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2)$:
   (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
       i. $i \xleftarrow{\$} \mathbb{Z}_q^*$;  if $i \in \mathit{Domain}(\pi)$ then abort
       ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
   (b) invoke $(\texttt{map}, \pi^{-1}(\mathcal{P}_1) + \pi^{-1}(\mathcal{P}_2))$ and return the result

4. To process a request to sign $m$:
   (a) $h \leftarrow \mathit{Hash}(m) \in \mathbb{Z}_q$
   (b) $r \xleftarrow{\$} \mathbb{Z}_q^*$;  if $r^* \in \mathit{Domain}(\pi)$ then abort
   (c) invoke $(\texttt{map}, r)$ to get $\mathcal{R}$
   (d) $t \leftarrow \bar{C}(\mathcal{R}) \in \mathbb{Z}_q$
   (e) if $t = 0$ then abort
   (f) if $h + td = 0$ then abort
   (g) $s \leftarrow r^{-1}(h + td)$
   (h) return $(\mathcal{R}, s, t)$
```

The changes are highlighted. It is trivial to verify that the adversary's forging advantage in this game differs from that in the original attack game by $O(N^2/q)$. Indeed, we can view both games as operating on the same sample space, and both games proceed identically unless a specific failure event occurs in the Lazy-Sim1-based game. One sees that this failure event occurs with probability $O(N^2/q)$ — here we make use of the fact (among others) that t has a guessing probability of $O(1/q)$.

Now we modify the logic for processing signing requests, as follows:

```
4. To process a request to sign $m$:
   (a) $h \leftarrow \mathit{Hash}(m) \in \mathbb{Z}_q$
   (b) $\mathcal{R} \xleftarrow{\$} E^*$
   (c) if $\mathcal{R} \in \mathit{Range}(\pi)$ then abort
   (d) $t \leftarrow \bar{C}(\mathcal{R}) \in \mathbb{Z}_q$
   (e) if $t = 0$ then abort
   (f) $s \xleftarrow{\$} \mathbb{Z}_q^*$
   (g) $r \leftarrow s^{-1}(h + td)$
   (h) if $r \in \mathit{Domain}(\pi)$ then abort
   (i) add $(-r, -\mathcal{R})$ and $(r, \mathcal{R})$ to $\pi$
   (j) return $(\mathcal{R}, s, t)$
```

Let us call this **Lazy-Sim2**. It is easy to verify that this *perfectly* simulates the behavior of Lazy-Sim1, and so the adversary's forgery advantage does not change at all. Indeed, both simulators choose $\mathcal{R}$ at random and abort if $\mathcal{R} \in \mathit{Range}(\pi)$ or $t \coloneqq \bar{C}(\mathcal{R})$ satisfies $t = 0$ or $h + td = 0$. Moreover, if they do not abort, then both will generate r at random and abort if $r \in \mathit{Domain}(\pi)$. So, in processing a signing request, the probability that the signing request aborts is the same in both simulations, and the distribution of $(r, \mathcal{R})$ given that it does not abort is identical.

We now define a **symbolic** simulation of the attack Game. The essential difference in this game is that $\mathit{Domain}(\pi)$ will now consist of polynomials of the form $a + b\mathtt{D}$, where $a, b \in \mathbb{Z}_q$ and $\mathtt{D}$ is an indeterminant. Note that π will otherwise still satisfy all of the requirements of an encoding function. The simulator is identical to Lazy-Sim2, except has highlighted:

Symbolic-Sim:

```
1. Initialization:
    (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
    (b) $d \xleftarrow{\$} \mathbb{Z}_q^*$
    (c) invoke $(\mathtt{map}, 1)$ to obtain $\mathcal{G}$
    (d) invoke $(\mathtt{map}, \mathtt{D})$ to obtain $\mathcal{D}$
    (e) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\mathtt{map}, i)$:
    (a) if $i \notin \mathit{Domain}(\pi)$:
        i. $\mathcal{P} \xleftarrow{\$} E^*$;  if $\mathcal{P} \in \mathit{Range}(\pi)$ then abort
        ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
    (b) return $\pi(i)$
3. To process a group oracle query $(\mathtt{add}, \mathcal{P}_1, \mathcal{P}_2)$:
    (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
        i. $i \xleftarrow{\$} \mathbb{Z}_q^*$;  if $i \in \mathit{Domain}(\pi)$ then abort
        ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
    (b) invoke $(\mathtt{map}, \pi^{-1}(\mathcal{P}_1) + \pi^{-1}(\mathcal{P}_2))$ and return the result
4. To process a request to sign $m$:
    (a) $h \leftarrow \mathit{Hash}(m) \in \mathbb{Z}_q$
    (b) $\mathcal{R} \xleftarrow{\$} E^*$
    (c) if $\mathcal{R} \in \mathit{Range}(\pi)$ then abort
    (d) $t \leftarrow \bar{C}(\mathcal{R}) \in \mathbb{Z}_q$
    (e) if $t = 0$ then abort
    (f) $s \xleftarrow{\$} \mathbb{Z}_q^*$
    (g) $r \leftarrow s^{-1}(h + t\mathtt{D})$
    (h) if $r \in \mathit{Domain}(\pi)$ then abort
```

```
        (i) add $(-r, -\mathcal{R})$ and $(r, \mathcal{R})$ to $\pi$
        (j) return $(\mathcal{R}, s, t)$
```

We can model the attack game with respect to simulators Lazy-Sim2 and Symbolic-Sim as operating in the same underlying sample space, comprising

- the random tape of the adversary,
- the random choice of d,
- the random choices if i in Step 3(b)(i).

That is, the outcomes of both games are determined by these values in the sample space, although the computations performed in each game are different, and so the outcomes of the games may differ.

Let us define the following Event Z, which we defined in terms of the Symbolic-Sim-based attack game. For a polynomial P in $\mathbb{Z}_q[\mathtt{D}]$, we define $[P] \in \mathbb{Z}_q$ to be the value of P with $\mathtt{D}$ replaced by d. For a set S of such polynomials, we define $[S] := \{[P] : P \in S\}$. Event Z is the event that at one of the highlighted tests of the form "$P \in \mathit{Domain}(\pi)$", we have $P \notin \mathit{Domain}(\pi)$ but $[P] \in [\mathit{Domain}(\pi)]$.

We claim that these two games proceed identically unless Z occurs. This should be clear. It follows the forging probability in these two games differs by at most $\Pr[Z]$.

It should also be clear from the Schwartz-Zippel Lemma that $\Pr[Z] = O(N^2/q)$. Here, we use the fact that in the Symbolic-Sim-based attack game, the value of d is independent of the coefficients of the polynomials that determine Event Z.

Note that Symbolic-Sim as defined here is identical to Symbolic-Sim defined in Section 5.2, except that in the latter, we have deleted the initialization of d in Step 1(b) of the former, as d is not actually needed. That proves the lemma.

B Proof of Lemma 2

We start by modifying Lazy-Sim as follows.

Lazy-Sim1:

```
1. Initialization:
    (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
    (b) $d \overset{\$}{\leftarrow} \mathbb{Z}_q^*$
    (c) invoke $(\mathtt{map}, 1)$ to obtain $\mathcal{G}$
    (d) invoke $(\mathtt{map}, d)$ to obtain $\mathcal{D}$
    (e) $k \leftarrow 0$;  $K \leftarrow \emptyset$
    (f) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\mathtt{map}, i)$:
    (a) if $i \notin \mathit{Domain}(\pi)$:
        i. $\mathcal{P} \overset{\$}{\leftarrow} E^*$;  if $\mathcal{P} \in \mathit{Range}(\pi)$ then abort
```

```
            ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
        (b) return $\pi(i)$

    3. To process a group oracle query $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2)$:
        (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
            i. $i \overset{\$}{\leftarrow} \mathbb{Z}_q^*$;  if $i \in \mathit{Domain}(\pi)$ then $\texttt{abort}$
            ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
        (b) invoke $(\texttt{map}, \pi^{-1}(\mathcal{P}_1) + \pi^{-1}(\mathcal{P}_2))$ and return the result

    4. To process a presignature request:
        (a) $k \leftarrow k + 1$
        (b) $r_k \overset{\$}{\leftarrow} \mathbb{Z}_q^*$;  if $r_k \in \mathit{Domain}(\pi)$ then $\texttt{abort}$
        (c) invoke $(\texttt{map}, r_k)$ to get $\mathcal{R}$
        (d) $t_k \leftarrow \bar{C}(\mathcal{R}_k) \in \mathbb{Z}_q$
        (e) if $t_k = 0$ then $\texttt{abort}$
        (f) $K \leftarrow K \cup \{k\}$; return $\mathcal{R}_k$

    5. To process a request to sign $m_k$ using presignature number $k \in K$:
        (a) $K \leftarrow K \setminus \{k\}$
        (b) $h_k \leftarrow \mathit{Hash}(m_k) \in \mathbb{Z}_q$
        (c) if $h_k + t_k d = 0$ then return $\mathit{fail}$
        (d) $s_k \leftarrow r_k^{-1}(h_k + t_k d)$
        (e) return $(s_k, t_k)$
```

The changes are highlighted. It is trivial to verify that the adversary's forging advantage in this game differs from that in the original attack game by $O(N^2/q)$.

We then modify this simulator as follows.

Lazy-Sim2:

```
    1. Initialization:
        (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
        (b) $d \overset{\$}{\leftarrow} \mathbb{Z}_q^*$
        (c) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
        (d) invoke $(\texttt{map}, \texttt{D})$ to obtain $\mathcal{D}$
        (e) $k \leftarrow 0$; $K \leftarrow \emptyset$
        (f) return $(\mathcal{G}, \mathcal{D})$

    2. To process a group oracle query $(\texttt{map}, i)$:
        (a) if $i \notin \mathit{Domain}(\pi)$:
            i. $\mathcal{P} \overset{\$}{\leftarrow} E^*$;  if $\mathcal{P} \in \mathit{Range}(\pi)$ then $\texttt{abort}$
            ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
        (b) return $\pi(i)$
```

3. To process a group oracle query $(\mathtt{add}, \mathcal{P}_1, \mathcal{P}_2)$:
    (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
        i. $i \xleftarrow{\$} \mathbb{Z}_q^*$; if $i \in \mathit{Domain}(\pi)$ then $\mathtt{abort}$
        ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to π
    (b) invoke $(\mathtt{map}, \pi^{-1}(\mathcal{P}_1) + \pi^{-1}(\mathcal{P}_2))$ and return the result
4. To process a presignature request:
    (a) $k \leftarrow k + 1$
    (b) $r_k \xleftarrow{\$} \mathbb{Z}_q^*$;
    (c) invoke $(\mathtt{map}, \mathtt{R}_k)$ to get $\mathcal{R}_k$
    (d) $t_k \leftarrow \bar{C}(\mathcal{R}_k) \in \mathbb{Z}_q$
    (e) if $t_k = 0$ then $\mathtt{abort}$
    (f) $K \leftarrow K \cup \{k\}$; return $\mathcal{R}_k$
5. To process a request to sign m_k using presignature number $k \in K$:
    (a) $K \leftarrow K \setminus \{k\}$
    (b) $h_k \leftarrow \mathit{Hash}(m_k) \in \mathbb{Z}_q$
    (c) if $h_k + t_k d \neq 0$ then $s_k \leftarrow r_k^{-1}(h_k + t_k d)$ else $s_k \xleftarrow{\$} \mathbb{Z}_q^*$
    (d) substitute $s_k^{-1}(h_k + t_k\mathtt{D})$ for $\mathtt{R}_k$ throughout $\mathit{Domain}(\pi)$, and $\mathtt{abort}$ if any two polynomials collapse
    (e) return (s_k, t_k)

Again, changes are highlighted. We can view these last two games as operating on the same sample space. For a polynomial P in the variables $\mathtt{D}, \mathtt{R}_1, \mathtt{R}_2, \ldots$, let $[P]$ denote the value obtained by substituting $\mathtt{D} \leftarrow d, \mathtt{R}_1 \leftarrow r_1, \mathtt{R}_2 \leftarrow r_2, \ldots$. For a set S of such polynomials, we define $[S] \coloneqq \{[P] : P \in S\}$. Let Z be the event that in the Lazy-Sim2 game that either

- $h_k + t_k d = 0$ for some signing request k, or
- at one of the highlighted tests of the form "$P \in \mathit{Domain}(\pi)$", we have $P \notin \mathit{Domain}(\pi)$ but $[P] \in [\mathit{Domain}(\pi)]$.

Suppose we are processing a signing request using the kth presignature, and that Z has not been triggered before this signing request and $h_k + t_k d \neq 0$. Then:

- when the subsitution in Step 5(d) is attempted, the values $[P]$ for $P \in \mathit{Domain}(\pi)$ are all distinct;
- in particular, the substitution will not fail.

We claim that:

1. these last two games proceed identically unless Z occurs;
2. $\Pr[Z] = O(N^2/q)$.

These two claims together imply that the adversary's advantage in these two games differs by $O(N^2/q)$.

The second claim essentially follows from the Schwartz-Zippel Lemma, and the following observations:

- at any point in time in the Lazy-Sim2 game, if the polynomials in $\mathit{Domain}(\pi)$ involve the variables $\mathtt{D}$ and $\{\mathtt{R}_k\}_{k \in K}$, then the coefficients of these polynomials are independent of d and $\{r_k\}_{k \in K}$.

One can then verify that this last game is completely equivalent to the Symbolic-Sim-based game in Section 7.2.

C Wagner's algorithm

We sketch Wagner's algorithm, adapted to our setting. Let $n \approx \log_2(q)$ be the number of bits of q. Let us assume that we are given a list of random h values and a list of random h^* values, where these two lists are each of size 2^ℓ. Similarly, we are given a list of random e values and a list of random e^* values, where these two lists are each of size $2^{\ell'}$. We want to find h, h^*, e, e^* such that

$$h - e = h^* - e^*. \tag{10}$$

Here, we have absorbed the value t into the e and e^* values, since it does not play a significant role in this analysis.

Let us assume that $\ell' \leq n/3$ and set ℓ so that

$$\ell \approx \frac{n - \ell'}{2} \geq \ell'.$$

Here, we are assuming that while the set of possible key tweaks may be bounded, the adversary has complete control over how many messages it may hash. Wagner's algorithm represents elements of $\mathbb{Z}_q$ as a balanced residue, in the interval $[-q/2, +q/2]$, expressed using n-bit two's complement notation. The first step of the algorithm finds the set S of all pairs (h, e) such that h and e agree in their high order ℓ' bits. Similarly, it finds the set S^* all pairs (h^*, e^*) such that h^* and e^* agree in their high order ℓ' bits. The expected size of S and S^* is roughly

$$2^\ell \cdot 2^{\ell'} / 2^{\ell'} = 2^\ell,$$

and the time spent finding them (using standard hashing techniques) is $O(2^\ell)$.

For each pair $(h, e) \in S$, the value $h - e$ is a random number in a set S of size about $2^{n-\ell'}$. Similarly for the each pair (h^*, e^*). The expected number of $(h, e) \in S$ and $(h^*, e^*) \in S^*$ such that $h - e = h^* - e^*$ is roughly

$$2^{2\ell} / 2^{n-\ell'} = 2^{2\ell - n + \ell'} \geq 1.$$

So we expect that are such $(h, e) \in S$ and $(h^*, e^*) \in S^*$, and we can find them (using standard hashing techniques) in time $O(2^\ell)$.

So we can find h, h^*, e, e^* satisfying (10) in time $O(2^\ell)$, where $\ell \approx (n - \ell')/2$. Thus, we see that as ℓ' grows, we can beat the birthday attack. In particular, if $\ell' \approx n/3$, then we get an algorithm that runs in time $O(2^{n/3})$. It depends on the setting as to whether the set of valid tweaks can feasibly be as large as $2^{n/3}$, but if it is, we have a good chance of computing a forgery in time $O(2^{n/3})$.

D Relation to BIP32

In additive key derivation, we add a "tweak" $e \in \mathbb{Z}_q$. As we have described it, such a tweak e is chosen from some predetermined set $\mathfrak{E}$ of bounded size, or from a hash function Hash' applied to an identifier id. Of course, we could also insist on both: that e is derived by applying a hash to a predetermined set of identifiers of bounded size.

In this appendix, we discuss how the BIP32 standard [Wui20] corresponds to these assumptions.

We first review the BIP32 standard, presented with somewhat different notation and emphasis. BIP32 makes use of the curve `secp256k1` in [Cer10]. This is a curve of prime order q, where q is of the form

$$q = 2^{256} - q',$$

where

$$0 \le q' < 2^{129}.$$

Because of the special form of q, a randomly chosen integer in the range $[0, 2^{256})$ will lie outside the range $[0, q)$ with probability at most $2^{-(256-129)} = 2^{-127}$.

BIP32 makes use of HMAC-SHA512, which we denote here simply by HMAC. The function HMAC takes two inputs:

- the first input is the "key";
- the second input is the "data".

In general, both inputs are byte strings of arbitrary length. HMAC produces 64-byte outputs. It is based on a Merkle-Damgård design with a chaining variable 64 bytes, and a block size of 128 bytes.

Although HMAC was initially designed as a pseudo-random function, it is often assumed to be a random oracle (viewing *both* inputs as inputs to the oracle). [DRST13] show that HMAC is indifferentiable from a random oracle provided the set of keys is mildly restricted. Indeed, as shown in [DRST13], if an application only uses only fixed length keys of length at most 127 bytes, then HMAC is essentially as good as a random oracle. As we will see, BIP32 satisfies this restriction.

Some notation:

- Let B be the set of all bytes.
- Let $\mathcal{S} := B^*$, the set of all byte strings.
- Let $\mathcal{C} := B^{32}$, the set of all **chain codes**.
- Let HMAC_2 be the function that outputs (a, b), where a is the first 32 bytes of HMAC and b is the last 32 bytes.

- For $s \in \mathcal{S}$, let $[s]$ denote the integer for which s is a base-256 representation, and let $[s]_q$ denote the image of $[s]$ in $\mathbb{Z}_q$.
- For a group element $\mathcal{P} \in E$, let $\langle \mathcal{P} \rangle \in \mathcal{S}$ be the compressed SEC1 encoding of $\mathcal{P}$ [Cer09] — note that this is a prefix-free encoding.

For a group element $\mathcal{D} \in E$, let us define the function

$$\begin{array}{rrcl} H_{\mathcal{D}}: & (\mathbb{Z}_q \times \mathcal{C}) \times \mathcal{S} & \rightarrow & \mathbb{Z}_q \times \mathcal{C} \\ & (\ (e,c),\ s\ ) & \mapsto & (e + [a]_q, b),\ \text{where}\ (a,b) := \mathrm{HMAC}_2\big(\ c,\ \langle \mathcal{D} + e\mathcal{G} \rangle \parallel s\ \big). \end{array}$$

We then define

$$H^*_{\mathcal{D}}: \ \mathcal{S}^* \ \rightarrow \ \mathbb{Z}_q \times \mathcal{C}$$

as follows:

$$H^*_{\mathcal{D}}(s_1, \ldots, s_\ell) := \begin{cases} (0, \mathtt{IV}) & \text{if } \ell = 0, \\ H_{\mathcal{D}}\big(\ H^*_{\mathcal{D}}(s_1, \ldots, s_{\ell-1}),\ s_\ell\ \big) & \text{if } \ell > 0. \end{cases}$$

Here, IV is an arbitrary, fixed element of $\mathcal{C}$. Let $H^+_{\mathcal{D}}$ be $H^*_{\mathcal{D}}$ restricted to the domain $\mathcal{S}^+$.

This is essentially the BIP32 derivation function. The main difference is that in BIP32, the function $H_{\mathcal{D}}$ will *fail* if $[a] \geq q$ or $(e + [a]_q)\mathcal{G} + \mathcal{D} = \mathcal{O}$. Modeling HMAC as a random oracle, and because of the special form of q, this failure will occur with negligible probability, so we can safely ignore such failures. The only other difference is that in BIP32, the byte strings $s_1, \ldots, s_\ell$ are restricted to being exactly 4 bytes long, but this is not essential for any security properties (since the compressed SEC1 encoding is prefix free).

Given a master public key $\mathcal{D} \in E$ and $(s_1, \ldots, s_\ell) \in \mathcal{S}^*$, if $H^*_{\mathcal{D}}(s_1, \ldots, s_\ell) = (e, c)$, then $\mathcal{D} + e\mathcal{G}$ is the corresponding public key derived from $\mathcal{D}$ via $(s_1, \ldots, s_\ell)$. If $d \in \mathbb{Z}_q$ is the master secret key, so that $\mathcal{D} = d\mathcal{G}$, then $d + e$ is the corresponding derived secret key. Observe that any party who knows $\mathcal{D}$ and $(s_1, \ldots, s_\ell)$ (as well as IV) can compute the derived public key. Note that some use cases of BIP32 consider IV to be private. One such use case is where users want derived public keys to be unlinkable. However, our analysis here assumes it is public, which is sufficient for analyzing the security of ECDSA against forgery using derived public keys.

It would be nice to show that $H^+_{\mathcal{D}}$ is indifferentiable from a random oracle (RO), in the sense defined in [CDMP05]. However, because of extension attacks, we cannot hope to do this. However, we can still show that $H^+_{\mathcal{D}}$ is indifferentiable from a so-called **public-use random oracle (pub-RO)**, a notion defined in [DRS09]. Essentially, with a pub-RO, the adversary is allowed to ask for all queries made to the random oracle by any honest parties. This is sufficient for analyzing the security of ECDSA with tweaks derived via $H^+_{\mathcal{D}}$, as their are no "secret" inputs to $H^+_{\mathcal{D}}$ made by the challenger in the forgery attack game.

It is shown in Theorem 7.1 in [DRS09] that the Merkle-Damgård construction applied to a pub-RO compression function is itself indifferentiable from a pub-RO. This theorem does not require any "strengthening" (i.e., a suffix-free encoding of the input).

Since $H^+_{\mathcal{D}}$ is exactly the Merkle-Damgård construction applied to the compression function $H_{\mathcal{D}}$, we could apply this result here, provided we can show that $H_{\mathcal{D}}$ is indifferentiable from a pub-RO, where HMAC is modeled as a random oracle. Unfortunately, we cannot do this without computing discrete logs. Indeed, given an input $(c, \langle \mathcal{P} \rangle \parallel s)$ to HMAC, an indifferentiabilty simulator would have to be able to determine $e \in \mathbb{Z}_q$ such that $\mathcal{P} = \mathcal{D} + e\mathcal{G}$, and query the oracle representing $H_\mathcal{D}$ at the point $((e, c), s)$, just in case the adversary would later query $H_\mathcal{D}$ at this point.

Luckily for us, Theorem 7.1 in [DRS09] actually applies to any compression function that is indifferentiable from what [DRS09] call a **public-use *guarded* random oracle (pub-GRO)**. Roughly speaking, in the pub-GRO indifferentiabilty game for $H_\mathcal{D}$, the adversary does not have unfettered access to the oracle representing $H_\mathcal{D}$, but only to an inputs of the form $((e, c), s)$, where (e, c) is an **allowable pair** in the following sense: either $(e, c) = (0, \mathtt{IV})$ or (e, c) previously output by the oracle. This restriction avoids the problem indicated above. Indeed, given an input $(c, \langle \mathcal{P} \rangle \parallel s)$ to HMAC, an indifferentiabilty simulator can test if there is an allowable pair of the form (e, c) where $\mathcal{P} = \mathcal{D} + e\mathcal{G}$; if not, we can safely assume that $H_\mathcal{D}$ will never be queried at the corresponding point, and so the simulator can just respond with random junk.

We claim the following: *assuming* HMAC *is modeled as a random oracle, $H_\mathcal{D}$ is indifferentiable from a pub-GRO.* To prove this, one must take into account the special form of q, which ensures that the uniform distribution on $\{0, \ldots, q-1\}$ is statistically very close to the uniform distribution on $\{0, \ldots, 2^{256} - 1\}$. One must also make use of the fact that the compressed SEC1 encoding is prefix free. From this, the claim follows.

From the claim, we may conclude that $H_\mathcal{D}^+$ is indifferentiable from a pub-RO.

Now consider the function

$$\begin{array}{rccc} \pi_1 : & \mathbb{Z}_q \times \mathcal{C} & \to & \mathbb{Z}_q \\ & (e, c) & \mapsto & e, \end{array}$$

which projects onto its first argument. The function we are ultimately interested in is $Hash'_\mathcal{D} := \pi_1 \circ H_\mathcal{D}^*$. This function maps a variable length tuple $(s_1, \ldots, s_\ell) \in \mathcal{S}^*$ to a tweak $e \in \mathfrak{E}$. By the above observations, $Hash'_\mathcal{D}$ restricted to $\mathcal{S}^+$ is indifferentiable from a pub-RO (and $Hash'_\mathcal{D}() = 0$).

It is also easy to show that $Hash'_\mathcal{D}$ is collision resistant, assuming that the function $\pi_1 \circ H_\mathcal{D}$ is collision resistant and that it is hard to find a preimage of 0 under $\pi_1 \circ H_\mathcal{D}$ (the latter condition is needed, since we are not using any "strengthening" in the Merkle-Damgård construction).

References

[BB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In P. Rudnicki, editor, *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989*, pages 201–209. ACM, 1989. doi:10.1145/72981.72995.

[BLN+09] D. J. Bernstein, T. Lange, R. Niederhagen, C. Peters, and P. Schwabe. Implementing Wagner's generalized birthday attack against the SHA-3 round-1 candidate FSB. Cryptology ePrint Archive, Report 2009/292, 2009. `https://ia.cr/2009/292`.

[BLS01] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001. doi:10.1007/3-540-45682-1_30.

[Bro02] D. R. L. Brown. Generic groups, collision resistance, and ECDSA. *Designs, Codes and Cryptography*, 35:119–152, 2002.

[CDMP05] J. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005. doi:10.1007/11535218_26.

[Cer09] Certicom Research. Sec 1: Elliptic curve cryptography, 2009. Version 2.0, `http://www.secg.org/sec1-v2.pdf`.

[Cer10] Certicom Research. Sec 2: Recommended elliptic curve domain parameters, 2010. Version 2.0, `http://www.secg.org/sec2-v2.pdf`.

[CMP20] R. Canetti, N. Makriyannis, and U. Peled. UC non-interactive, proactive, threshold ECDSA. Cryptology ePrint Archive, Report 2020/492, 2020. `https://ia.cr/2020/492`.

[DEF$^+$21] P. Das, A. Erwig, S. Faust, J. Loss, and S. Riahi. The exact security of BIP32 wallets. Cryptology ePrint Archive, Report 2021/1287, 2021. `https://ia.cr/2021/1287`.

[DFI22] The DFINITY Team. The internet computer for geeks. Cryptology ePrint Archive, Report 2022/087, 2022. `https://ia.cr/2022/087`.

[DJN$^+$20] I. Damgård, T. P. Jakobsen, J. B. Nielsen, J. I. Pagter, and M. B. Østergård. Fast threshold ECDSA with honest majority. Cryptology ePrint Archive, Report 2020/501, 2020. `https://ia.cr/2020/501`.

[DRS09] Y. Dodis, T. Ristenpart, and T. Shrimpton. Salvaging Merkle-Damgård for practical applications. In A. Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2009. doi:10.1007/978-3-642-01001-9_22.

[DRST13] Y. Dodis, T. Ristenpart, J. Steinberger, and S. Tessaro. To hash or not to hash again? (in)differentiability results for H^2 and HMAC. Cryptology ePrint Archive, Report 2013/382, 2013. `https://eprint.iacr.org/2013/382`.

[FKP16] M. Fersch, E. Kiltz, and B. Poettering. On the provable security of (EC)DSA signatures. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1651–1662. ACM, 2016. doi:10.1145/2976749.2978413.

[GG20] R. Gennaro and S. Goldfeder. One round threshold ECDSA with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. `https://ia.cr/2020/540`.

[GS14] G. Gutoski and D. Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. Cryptology ePrint Archive, Report 2014/998, 2014. `https://ia.cr/2014/998`.

[Nec94] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994. Translated from *Matematicheskie Zametki*, 55(2):91–101, 1994.

[NIST13] National Institute of Standards and Technology. Digital signature standard (DSS). Federal Information Processing Publication 186-4, 2013. `https://doi.org/10.6028/NIST.FIPS.186-4`.

[NS15] I. Nikolic and Y. Sasaki. Refinements of the k-tree algorithm for the generalized birthday problem. In T. Iwata and J. H. Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 683–703. Springer, 2015. doi:10.1007/978-3-662-48800-3_28.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997. doi:10.1007/3-540-69053-0_18.

[SPMS02] J. Stern, D. Pointcheval, J. Malone-Lee, and N. P. Smart. Flaws in applying proof methodologies to signature schemes. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 93–110. Springer, 2002. doi:10.1007/3-540-45708-9_7.

[Wag02] D. A. Wagner. A generalized birthday problem. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002. doi:10.1007/3-540-45708-9_19.

[Wui20] P. Wuille. Hierarchical deterministic wallets, 2020. `https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`.

[YY19] T. H. Yuen and S. Yiu. Strong known related-key attacks and the security of ECDSA. In J. K. Liu and X. Huang, editors, *Network and System Security - 13th International Conference, NSS 2019, Sapporo, Japan, December 15-18, 2019, Proceedings*, volume 11928 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2019. doi:10.1007/978-3-030-36938-5_8.