On the Impossibility of Short Algebraic Signatures

Nico Döttling[1], Dominik Hartmann[2], Dennis Hofheinz[3], Eike Kiltz[2], Sven Schäge[2], and Bogdan Ursu[3]

[1] CISPA Saarbrücken
nico.doettling@gmail.com
[2] Ruhr-University Bochum
dominik.hartmann@rub.de, eike.kiltz@rub.de, sven.schaege@rub.de
[3] Department of Computer Science, ETH Zurich
hofheinz@inf.ethz.ch, bogdan.ursu@inf.ethz.ch

Abstract. The existence of one-way functions implies secure digital signatures, but not public-key encryption (at least in a black-box setting). Somewhat surprisingly, though, *efficient* public-key encryption schemes appear to be much easier to construct from concrete algebraic assumptions (such as the factoring of Diffie-Hellman-like assumptions) than efficient digital signature schemes. In this work, we provide one reason for this apparent difficulty to construct efficient signature schemes.

Specifically, we prove that a wide range of *algebraic* signature schemes (in which verification essentially checks a number of linear equations over a group) fall to conceptually surprisingly simple linear algebra attacks. In fact, we prove that in an algebraic signature scheme, sufficiently many signatures can be linearly combined to a signature of a fresh message. We present attacks both in known-order and hidden-order groups (although in hidden-order settings, we have to restrict our definition of algebraic signatures a little). More explicitly, we show:

- the insecurity of all signature schemes in Maurer's generic group model (in pairing-free groups), as long as the signature schemes do not rely on other cryptographic assumptions, such as hash functions.
- the insecurity of a natural class of signatures in hidden-order groups, where verification consists of linear equations over group elements.

We believe that this highlights the crucial role of *public* verifiability in digital signature schemes. Namely, while public-key encryption schemes do not require any publicly verifiable structure on ciphertexts, it is exactly this structure on signatures that invites attacks like ours and makes it hard to construct efficient signatures.

1 Introduction

Digital signatures and public-key encryption. Digital signatures and public-key encryption (PKE) schemes are two of the most fundamental cryptographic primitives. Both of them are crucial to securing communication, and are used in countless applications. From a theoretical perspective, these primitives are comparable, but somewhat different in strength: it is known that existentially, digital signatures are equivalent to one-way functions [30,31]. (That is, secure digital signatures can be constructed from one-way functions and vice versa.) However, PKE schemes appear to be strictly stronger (in a black-box sense) than one-way functions [22]. In this sense, it is easier to construct digital signatures than PKE schemes.

On the other hand, current *efficient* constructions of signatures and PKE schemes from stronger (in particular group-based) assumptions paint a different picture. For instance, efficient PKE schemes are known from the factoring [20] and DDH [9,24] assumptions. But *efficient* signature schemes from the factoring or DDH assumptions appear to require random oracles [32,5,23], stronger assumptions [10], or tradeoffs in efficiency [21]. Hence, it seems that efficient signatures are somewhat harder to construct than PKE schemes. This leads to the obvious question that motivates this work:

What makes efficient (standard-model) digital signature schemes harder to construct than PKE schemes?

We note that while lower bounds for digital signature schemes exist, they are either limited to very special types of signature schemes (like structure-preserving signatures [2,1,16,17] in the pairing setting), or to bounds on the efficiency of constructions from symmetric primitives [14,4]. To the best of our knowledge, e.g., the (space or time) complexity of group-based signature schemes (without pairings) is not well-understood.

The role of public verifiability. Of course, signature and PKE schemes have a very different syntax (and different goals). However, one unique property of digital signatures is that of *public verifiability*. Specifically, even an adversary can verify the validity of a signature, while it can in general not verify the consistency of a PKE ciphertext (i.e., that it was generated with the encryption procedure). In a security reduction, this difference allows a wider range of techniques to modify ciphertexts in a security reduction than signatures. In fact, a popular technique for PKE security reductions starts by making the ciphertext inconsistent [9,11,24]. Similar techniques for signatures exist [15], but currently require a very specific algebraic setup in order to be compatible with public verifiability.

In a nutshell, public verifiability enforces a certain publicly verifiable structure on signatures that does not need to be present in PKE ciphertexts. For many known signature schemes (e.g., [12,32,10,6,35,21]), verifying this structure amounts to checking whether different group elements or exponents stored in the signature fulfill one or more polynomial equations (whose coefficients are derived from public key, message, and signature). The simpler these equations, the more efficient the signature scheme becomes.

Our results. We provide impossibility results in two settings:

– In pairing-free groups of known order, we show that all signature schemes in Maurer's generic group model are insecure, as long as these signature

schemes do not rely on other cryptographic assumptions, such as random oracles.
- In hidden-order groups, we establish the insecurity of a natural class of signatures, where verification consists of linear equations over group elements. As a further extension, we show that BLS signatures are insecure when the BLS hash function is instantiated with a specific type of programmable hash function, such as Waters' programmable hash function used in his signature scheme [35].

In this work, we show that if the equations that determine when a signature is valid are linear, with coefficients that are "too predictable" (in a sense to be defined), then the signature scheme is insecure. In particular, we prove that in this case, sufficiently many known signatures for random messages can be linearly combined to form a new signature for a fresh message.

As a simple special case, assume a signature scheme for which valid signatures $(\sigma_1, \ldots, \sigma_k)$ satisfy a single equation of the form

$$\sigma_1 P_1 + \ldots + \sigma_k P_k = P_0$$

over an additive group $(\mathbb{G}, +)$, where the $P_i = P_i(\mathsf{vk}, m) \in \mathbb{G}$ are publicly computable from verification key and message, and the σ_i are exponents from the signature.

Now if the P_i are also (publicly) computed in a linear fashion from a few base elements $X_1, \ldots, X_n \in \mathbb{G}$ in the verification key, each signature gives a linear equation

$$\sigma'_1 X_1 + \ldots + \sigma'_n X_n = 0$$

(with known σ'_i) for the X_i. After seeing sufficiently many valid signatures, the σ'_i (and hence the σ_i) for a fresh random message m^* can be derived by linear algebra. Hence, such a signature scheme is insecure.

Extensions. We extend this idea also to more equations, groups of unknown order (in which linear algebra has to be replaced with computations over the integers), and to randomized signatures. Our results cover typical settings of groups with known order (as used, e.g., with Diffie-Hellman-like assumptions), and with unknown order (as used, e.g., for factoring- and RSA-based constructions).

On the efficiency of our attacks. The basic attack outlined above is efficient in the sense that only linear algebra operations over matrices of exponents are made. When generalizing to unknown-order settings, these operations become linear algebra operations over the integers (which are more expensive, but remain polynomially efficient).

However, we will also generalize these ideas to schemes in which signatures contain arbitrary "tags" t (i.e., non-algebraic bitstrings that may influence the selection of the $P_i = P_i(\mathsf{vk}, m, t)$). Such tags can model, e.g., random coins chosen during signing. In general, it is not immediately clear how to adapt the above attack idea to such "tagged" signatures, since the attack only yields forgery coefficients σ_i, but no suitable t.

In this general setting, we give an attack that is "pseudo-efficient" in the generic group model. More specifically, our attack uses only a polynomial number of group operations, but brute-forces a suitable t (and thus becomes computationally infeasible for larger t). This attack shows that even for "tagged" signatures with such a t, security cannot come from hardness assumptions in the group alone. We stress that this generalized attack brute-forces only t, and becomes efficient also in terms of running time for logarithmically-short t. In fact, for empty t, it coincides with the basic attack described above.

What do our results say about existing paradigms for signature schemes? Our characterization also helps to understand what differentiates somewhat less efficient schemes like (implicitly [21] or explicitly [28]) tree-based schemes: in tree-based schemes, the polynomial equations checked have very diverse coefficients (in the sense that every message uses a unique set of coefficients). Our results do not apply in such settings, since we may end up with more variables than linear equations for these variables.

More applications. But while conceptually proving what *cannot* work to construct efficient signature schemes, we also showcase our techniques for a known signature scheme. Namely, we show that the pairing-based Boneh-Lynn-Shacham signature scheme [8], whose security is proved in the random oracle model, *cannot* be implemented with a suitable algebraic hash function (such as Waters' programmable hash function used in his signature scheme [35]).

1.1 Technical Outline

We will now provide a high level overview of our generic attacks. As purely combinatorial techniques suffice to achieve signatures of size logarithmic in the size of the message space [25,28,30], we will first suitably restrict the class of signature schemes under consideration.

Algebraic Signature Schemes. We will consider signature schemes which only make *algebraic* use of a cryptographic group. By algebraic, we mean that the group is only accessed via the standard group operations, but not by making use of representations of group elements. Essentially our notion of algebraic signature schemes is characterized by the property that verification checks a set of linear equations in the group. Specifically, assume in the following that we are working over a cyclic group $\mathbb{G}$ with generator P isomorphic to $\mathbb{Z}_p$. To simplify notation, we will write group operations additively. We say a signature scheme over an additive group $\mathbb{G}$ is algebraic, if it meets the following structural requirements.

- Signing keys sk are arbitrary bit-strings, whereas verification keys consist of a vector of n group elements $\mathbf{X} = (X_1, \ldots, X_n)^\intercal$ and a bit-string s.
- The signing algorithm produces signatures σ which consist of a vector of k group elements $\mathbf{Y} = (Y_1, \ldots, Y_k)^\intercal$ and a bit string t.

– The verification algorithm $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$ is described by two efficiently computable functions A and B, where $A(s, m, t)$ returns a $\mathbb{Z}_p^{\ell \times n}$ matrix and $B(s, m, t)$ returns a $\mathbb{Z}_p^{\ell \times k}$ matrix, respectively. The signature σ is accepted if the group-equations

$$A(s, m, t) \cdot \mathbf{X} = B(s, m, t) \cdot \mathbf{Y}$$

hold over the additive group $(\mathbb{G}, +)$.

We call this type of signature scheme *algebraic*, as the verification algorithm only makes algebraic use of the group, i.e. no bit-representations of group elements are used. Note further that this definition does not impose any restrictions on the signing algorithm, i.e. the signing algorithm may compute arbitrary functions of the signing key and the message. While at first glance this notion might seem overly restrictive, we will argue below that any signature scheme which only makes algebraic use of a group must be of this form. This notion does not include pairing-based constructions in the generic group model, since only linear verification equations are considered. Therefore constructions like [7,6] are not covered by our results.

Generic Group Models. Generic group models (GGMs) formalize the idea that algorithms, both schemes and adversaries, can only make algebraic use of a group, in other words only use the group *as a black box.* This is typically formalized by giving such algorithms access to the group via a *group oracle*, which takes as input $\mathbb{Z}_p$ elements and returns *handles* to group elements. The group oracle also performs group operations by taking handles of group elements and returning the handle of the resulting group element.

This idea can be implemented in different nuances. In Shoup's generic group model [34], the handles are chosen by a random injective function from $\mathbb{Z}_p$ into a set of sufficiently long bit-strings, i.e. each group element is represented by a unique but otherwise uniformly random handle. In this model, the group oracle can be immediately used to implement a random oracle [36]. As a consequence, in Shoup's generic group model Schnorr's signature scheme [32] (using a random oracle constructed from the group) is provably secure. But this means that in this model there do in fact exist fully succinct signature schemes, yet via a non-standard use of the group.

In Maurer's generic group model [26], the group oracle is stateful and handles are computed lazily via a counter. Consequently, the handles do not carry additional entropy and the group oracle cannot be used to implement a random oracle. Furthermore, it seems to be difficult to even define a consistent hash function on group elements, as the labels depend on the order in which the group elements were queried. In fact, in the technical part we argue that any signature scheme in Maurer's generic group model [26] must be algebraic as in the above sense.

Learning Linear Functions. We will now turn to showing that any algebraic signature scheme can be efficiently attacked in the generic group model, where we measure the adversary's efficiency only in terms of its group oracle queries. Our starting point is a basic fact about the learnability of linear functions. Consider an experiment where a challenger chooses some function F, and then plays the following game with an adversary $\mathcal{A}$. The challenger chooses an input $\mathbf{x}_i$ from some distribution $\mathcal{X}$ and for each $\mathbf{x}_i$, $\mathcal{A}$ can either decide to see $F(\mathbf{x}_i)$ or provide a guess for $F(\mathbf{x}_i)$, where in the latter case if the adversary guesses correctly it wins. Clearly, for general functions F this experiment is hopeless for the adversary, as F could, e.g., be a pseudorandom function with large output domain.

On the other hand, things are different if F is a linear function. Say $F : \mathsf{V} \to \mathsf{U}$ is a linear function between two vector spaces V and U over some field $\mathbb{F}$, where V is (say) of dimension n. Now, every time $\mathcal{A}$ is given a new input $\mathbf{x}_i \in \mathsf{V}$, then one out of two things must happen.

1. It holds that $\mathbf{x}_i$ is in the span of $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$. In this case, it follows from the linearity of F that $F(\mathbf{x}_i)$ is uniquely specified by the input-output pairs $(\mathbf{x}_1, F(\mathbf{x}_1)), \ldots, (\mathbf{x}_{i-1}, F(\mathbf{x}_{i-1}))$. Thus, in this case $\mathcal{A}$ can win the experiment with probability 1, simply by solving a linear equation system for $F(\mathbf{x}_i)$.
2. It holds that $\mathbf{x}_i$ is not in the span of $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$. In this case, $\mathcal{A}$ will learn new information about F.

Noting that the dimension of V is n, it follows that case 2 can happen at most n times. Consequently, after at most n rounds the adversary will win the experiment with probability 1.

Learning Affine Relations. The discussion in the previous paragraph does not immediately translate into an attack against algebraic signatures, as the signing algorithm is not necessarily a linear function. However, we will now modify the above argument such that it yields an efficient attack against algebraic signatures over prime order groups.

First note that for an algebraic signature scheme, knowing the discrete logarithms $\mathbf{x} = (x_1, \ldots, x_n)^\intercal$ of the group elements $\mathbf{X} = (X_1, \ldots, X_n)^\intercal$ is sufficient to forge signatures: Due to correctness of the signature scheme, we know that for any message m there exists a valid signature $\sigma = (\mathbf{Y}, t)$, i.e., it holds that

$$A(s, m, t) \cdot \mathbf{X} = B(s, m, t) \cdot \mathbf{Y}.$$

Consequently, it also holds

$$A(s, m, t) \cdot \mathbf{x} = B(s, m, t) \cdot \mathbf{y}.$$

in $\mathbb{Z}_p$. But this means that, given the discrete logarithms $\mathbf{x}$ of $\mathbf{X}$, we can find a signature for any message m by exhaustively searching over all possible values of t and testing for each t whether the equation system $B(s, m, t) \cdot \mathbf{y} = A(s, m, t) \cdot \mathbf{x}$ has a solution $\mathbf{y}$. If for a given t such a $\mathbf{y}$ exists, $(t, \mathbf{Y} = \mathbf{y}P)$ is a valid signature of the message m.

Towards developing the actual attack, we will now discuss a twist of the above idea. Assume the adversary already knows $(m_1, \sigma_1), \ldots, (m_{i-1}, \sigma_{i-1})$, where $\sigma_j = (\mathbf{Y}_j, t_j)$.

Then we can define a set $\mathsf{T}_{i-1} \subseteq \mathbb{Z}_p^n$ of *candidate vectors* $\mathbf{x}$ which could be the discrete logarithms of $\mathbf{X}$. A vector $\mathbf{x}$ is in T_{i-1}, if for all indices $j \in \{1, \ldots, i-1\}$ it holds that $A(s, m_j, t_j) \cdot \mathbf{X} = B(s, m_j, t_j) \cdot \mathbf{Y}_j$. In other words, T_{i-1} consists of all vectors $\mathbf{x}$ such that for $j \in \{1, \ldots, i-1\}$ the (m_j, σ_j) are valid message-signature pairs under the verification key $\mathbf{X}$.

Note that while membership in the set T_{i-1} can be decided using a polynomial number of group queries, we cannot efficiently compute the set T_{i-1} given the message-signature pairs $(m_1, \sigma_1), \ldots, (m_{i-1}, \sigma_{i-1})$. However observe that T_{i-1} is an affine subspace of $\mathbb{Z}_p^n$, as it is the solution-space of a non-homogenous linear equation system.

Ignoring the issue that we cannot efficiently compute T_{i-1} for the moment, we can now mount a similar learning argument as above. For an additional message-signature pair (m_i, σ_i), define T_i analogous to T_{i-1} taking the additional message-signature pair into account. So for every new message m_i, one out of two cases may happen:

1. By learning the signature σ_i of m_i the space T_i does not shrink, i.e., it holds that $\mathsf{T}_i = \mathsf{T}_{i-1}$.
2. By learning the signature σ_i the space T *does shrink*, i.e., $\mathsf{T}_i \subsetneq \mathsf{T}_{i-1}$.

Note that if the first case happens, an adversary $\mathcal{A}$ which knows T_{i-1} might have just as well computed σ_i *on its own*. Again ignoring the issue that this can't be implemented with a polynomial number of group operations, the adversary could exhaustively search over all $\sigma = (\mathbf{Y}, t)$ and pick one for which $\mathsf{T}_i = \mathsf{T}_{i-1}$. In the second case however, since both T_i and T_{i-1} are affine spaces, the dimension of T_i must be strictly smaller than the dimension of T_{i-1}. Since the space $\mathsf{T}_0 = \mathbb{Z}_p^n$ has dimension n, case 2 can happen at most n times.

Impossibility of Algebraic Signatures against Generic Adversaries. We will now address the issue that in the above sketch computing the affine spaces T_i cannot be achieved with a polynomial number of group operations. Upon closer inspection, the above argument only hinges on the fact that the dimension of T_{i-1} is decreasing. Since T_i is an affine space, it can be expressed as the sum of any point in T_i and a linear space W_i. By standard linear algebra, it holds that W_i is the intersection of the kernels of the $A(s, m_j, t_j)$. Clearly, W_i has the same dimension as T_i, i.e., whenever the dimension of T_i decreases, the dimension if W_i decreases as well.

However, instead of looking at T_i or W_i we will look at the *dual space* of T_i, that is the set of all homogeneous linear equations satisfied by all elements in T_i.

Specifically, for a verification key $\mathsf{vk} = (\mathbf{X}, s)$, a message m and a signature $\sigma = (\mathbf{Y}, t)$ we define the space

$$\mathsf{K}(m, t) = \mathrm{LKer}(B(s, m, t)) \cdot A(s, m, t),$$

where $\mathrm{LKer}(B(s, m, t))$ is the left-kernel of $B(s, m, t)$.

Notice that for every $\mathbf{v} \in \mathsf{K}(m,t)$ we can write $\mathbf{v}^\intercal = \mathbf{w}^\intercal A(s,m,t)$ for a $\mathbf{w}^\intercal \in \mathrm{LKer}(B(s,m,t))$ and it holds that

$$\mathbf{v}^\intercal \cdot \mathbf{X} = \mathbf{w}^\intercal \cdot A(s,m,t)\mathbf{X} = \mathbf{w}^\intercal \cdot B(s,m,t)\mathbf{Y} = 0,$$

as $\mathbf{w}^\intercal \in \mathrm{LKer}(B(s,m,t))$. In the main body we show that $\mathsf{K}(m,t)$ precisely characterizes the linear constraints imposed on the unknown vector $\mathbf{x}$ by s, m and t, that is if it holds for all $\mathbf{v}^\intercal \in \mathsf{K}(m,t)$ that $\mathbf{v}^\intercal \cdot \mathbf{X} = 0$ then there exists a $\mathbf{Y}$ such that $A(s,m,t)\mathbf{X} = B(s,m,t)\mathbf{Y}$. We further define L_i to be the set of linear constraints imposed by all $(m_1,\sigma_1),\dots,(m_i,\sigma_i)$, that is

$$\mathsf{L}_i = \bigoplus_{j=1}^{i} \mathsf{K}(m_j,t_j),$$

where $\oplus$ denotes the direct sum of vector spaces[4]. Note that $\mathsf{K}(m_i,t_i)$ and hence the L_i can be efficiently computed from the bit-strings s and $(m_1,t_1),\dots,(m_i,t_i)$. While the space T_i of candidates for $\mathbf{x}$ potentially shrinks when we add a new message-signature pair, the space L_i of linear relations that must be satisfied by all the $\mathbf{x}$ grows. As before, we will distinguish two cases concerning a new message-signature pair (m_i,σ_i).

1. In the first case it holds that $\mathsf{L}_i = \mathsf{L}_{i-1}$, in other words it holds that $\mathsf{K}(m_i,t_i) \subseteq \mathsf{L}_{i-1}$
2. In the second case it holds that $\mathsf{L}_{i-1} \not\subseteq \mathsf{L}_i$, i.e. $\mathsf{K}(m_i,t_i)$ contains new linear relations about $\mathbf{X}$.

We can routinely argue as before via a simple dimension argument that case 2 can happen at most n times. On the other hand, if case 1 happens for some (m_i,t_i), we can now efficiently forge a signature as follows. If $\mathsf{K}(m_i,t_i) \subseteq \mathsf{L}_{i-1}$, then by the above discussion there exists a $\mathbf{Y}$ such that $A(s,m_i,t_i)\mathbf{X} = B(s,m_i,t_i)\mathbf{Y}$. But the critical observation now is that this is a linear equation system for which we can find a solution $\mathbf{Y} \in \mathbb{G}^k$ (which is guaranteed to exist by the above discussion) by e.g. computing a *weak left-inverse*[5] H of $B(s,m,t)$ and setting

$$\mathbf{Y} = H \cdot A(s,m_i,t_i)\mathbf{X}.$$

Since H can be efficiently computed from $B(s,m,t)$, we can obtain $\mathbf{Y}$ from $\mathbf{X}$ by applying a polynomial number of computable $\mathbb{Z}_p$ operations to $\mathbf{X}$.

Consequently, the final attack can be described as follows, defining the spaces L_i as above. Initialize $\mathsf{L}_0 = \{0\}$ and repeat for pairwise distinct messages $m_1,\dots,$ m_{n+1}. For message m_i, check if there exists t_i such that $\mathsf{K}(m_i,t_i) \subseteq \mathsf{L}_{i-1}$. If so, compute $\mathbf{Y}_i$ as above, set $\sigma_i = (\mathbf{Y}_i,t_i)$ and output the forge (m_i,σ_i). Otherwise,

[4] The direct sum of vector spaces is the set of all vectors in the ambient space which can be linearly combined from vectors in these spaces

[5] A weak left-inverse of a matrix B is a matrix H for which it holds that $BHB = B$. For any matrix B the weak left-inverse H can be efficiently computed e.g. via gaussian elimination.

query a signature $\sigma_i = (\mathbf{Y}_i, t_i)$ from the signing oracle, set $\mathsf{L}_i \leftarrow \mathsf{L}_{i-1} \oplus \mathsf{K}(m_i, t_i)$ and continue.

Notice that while this attack needs to brute-force over all choices of the t_i, it only makes a polynomial number of queries to the group oracle. In fact we will show the slightly stronger statement that even if the adversary only receives a fixed number of random message/signature pairs, the above attack will work with overwhelming probability.

Algebraic Signatures in Groups of Unknown Order. For the above attack, we've constructed an adversary which makes a polynomial number of queries to the group oracle, but is otherwise unbounded. We will now consider a more restricted class of algebraic signatures over groups of *unknown order* and provide a fully efficient attack against this class of signatures by using a tweak on the above ideas. In this setting, we will not model the group as a generic group but rather provide efficient attacks against a simplified variant of algebraic signatures in any group of unknown order. Inspecting the above attack, the only inefficient part of the attack is the exhaustive search over the signature component t. In our notion of simplified algebraic signatures we will therefore require that the signature consists only of the group elements $\mathbf{Y}$.

Furthermore, in the unknown group order setting, we will model the publicly computable matrices $A(\mathsf{vk}, m)$ and $B(\mathsf{vk}, m)$ used by the verification algorithm as integer matrices. For a technical reason, in our notion of simplified signatures we will also require that the matrix B only depends on vk, but not on m.

Clearly, if the group order is not known, we cannot immediately extend the above argument, as we have used linear algebra over fields to compute the spaces of linear relations L_i. Now assume that for a verification key vk the adversary is given message-signature pairs $(m_1, \mathbf{Y}_1), \ldots, (m_{Q_s}, \mathbf{Y}_{Q_s})$, i.e. from the view of the adversary the following linear relations hold over the group:

$$\begin{gathered} A(\mathsf{vk}, m_1) \cdot \mathbf{X} = B(\mathsf{vk}) \cdot \mathbf{Y}_1 \\ \vdots \\ A(\mathsf{vk}, m_{Q_s}) \cdot \mathbf{X} = B(\mathsf{vk}) \cdot \mathbf{Y}_{Q_s}. \end{gathered}$$

Noting that the $A(\mathsf{vk}, m_i)$ are integer matrices in $\mathbb{Z}^{\ell \times n}$, then if the number of signatures Q_s issued to the adversary is greater than $\ell \cdot n$, we will observe integer linear relations between the $A(\mathsf{vk}, m_i)$, i.e. there exist $\alpha_1, \ldots, \alpha_{Q_s} \in \mathbb{Z}$ such that

$$\sum_{i=1}^{Q_s} \alpha_i A(\mathsf{vk}, m_i) = 0.$$

Assuming that $\alpha_{Q_s} \neq 0$, we can express $\alpha_{Q_s} A(\mathsf{vk}, m_{Q_s})$ as

$$\alpha_{Q_s} A(\mathsf{vk}, m_{Q_s}) = - \sum_{i=1}^{Q_s - 1} \alpha_i A(\mathsf{vk}, m_i).$$

Note that if $\alpha_{Q_s} = 1$, we can in fact forge a signature of the message m_{Q_s} given the message-signatures pairs $(m_1, \mathbf{Y}_1), \dots, (m_{Q_s-1}, \mathbf{Y}_{Q_s-1})$ as follows. Computing

$$\mathbf{Y}^*_{Q_s} = -\sum_{i=1}^{Q_s-1} \alpha_i \mathbf{Y}_i,$$

it holds that

$$\begin{aligned} B(\mathsf{vk})\mathbf{Y}^*_{Q_s} &= -\sum_{i=1}^{Q_s-1} \alpha_i B(\mathsf{vk})\mathbf{Y}_i = -\sum_{i=1}^{Q_s-1} \alpha_i A(\mathsf{vk}, m_i)\mathbf{X} \\ &= \alpha_{Q_s} A(\mathsf{vk}, m_{Q_s})\mathbf{X} = A(\mathsf{vk}, m_{Q_s})\mathbf{X}. \end{aligned}$$

Our main effort in Section 4 is devoted to showing for a sufficiently large but poly-bounded Q_s there indeed do exist $\alpha_1, \dots, \alpha_{Q_s} \in \mathbb{Z}$ with $\alpha_{Q_s} = 1$ such that

$$\sum_{i=1}^{Q_s} \alpha_i A(\mathsf{vk}, m_i) = 0.$$

A particular challenge of establishing this is that the existence of a signature for m_{Q_s} only guarantees such a linear relation modulo N (where N is the unknown group order). But to implement our attacks we need such a linear relation over $\mathbb{Z}$. We will further show that such a linear relation can be efficiently found using integer linear algebra techniques.

2 Preliminaries

2.1 Notation

We denote the security parameter by λ and assume that all algorithms implicitly take 1^λ as an additional input. For $n \in \mathbb{N}$, we define the set $[n] := \{1, \dots, n\}$. For a finite set S, $s \xleftarrow{\$} S$ denotes sampling s uniformly at random from S. Similarly, we write $s \xleftarrow{\$} A(x)$ for the output of a probabilistic algorithm A on input x and fresh random coins, and $s \leftarrow A(x)$ for deterministic algorithms. A probabilistic algorithm is PPT or efficient, if its runtime is polynomial in the security parameter and its inputs. For a cyclic group $\mathbb{G}$ of order N with generator P, we write $\mathcal{G} = (\mathbb{G}, N, P)$. We write all groups in additive notation and assume that the bit length of N is in $\mathcal{O}(\lambda)$. Specifically, the multiplication of a matrix of exponents and a group element vector is defined in the natural way, i.e. for $M = (m_{i,j}) \in \mathbb{Z}_N^{n \times k}$ and $\mathbf{X} = (X_1, \dots, X_k)^\intercal \in \mathbb{G}^k$, we define $M \cdot \mathbf{X} := (X'_1, \dots, X'_n)^\intercal$ with $X'_i = \sum_{j=1}^{k} m_{i,j} X_j$ for $i \in [n]$.

We will also use symmetric pairing groups, which we denote as $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, P, e)$, where $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and both $\mathbb{G}, \mathbb{G}_T$ are of order N, with $\mathbb{G}$ being generated by P.

We denote column vectors as lowercase bold $\mathbf{x} \in \mathbb{Z}^n$. For a matrix $M \in \mathbb{Z}^{n \times k}$, we denote by $\mathbf{m}_{*i}$ the i^{th} column of M and $\mathbf{m}_{j*}^\intercal$ the j^{th} row of M. The free module generated by the columns of a matrix $M \in \mathbb{Z}^{n \times k}$ is defined as $\mathsf{ColumnSpace}(M) := \{\mathbf{a} \in \mathbb{Z}^n : \text{ there exists } \mathbf{c} \in \mathbb{Z}^k \text{ with } \mathbf{a} = \sum_{j=1}^{k} c_j \mathbf{m}_{*j}\}$. Alternatively, we can also see $\mathsf{ColumnSpace}$ as an integer lattice in $\mathbb{Z}^n$.

Let K be a field and V be a K-vector space. For a vector subspace U of V, we write $\mathsf{U} \subseteq \mathsf{V}$. For a (finite) set of vector subspaces $\mathsf{U}_i \subseteq \mathsf{V}$ for $i \in I$, we denote the *direct sum* as $\mathsf{U} = \bigoplus_{i \in I} \mathsf{U}_i$, i.e. the smallest vector subspace $\mathsf{U} \subseteq \mathsf{V}$ s.t. $\mathsf{U}_i \subseteq \mathsf{U}$ for $i \in I$. Vectors of group elements are bold, upper case letters and vectors of group exponents are bold, lower case letters. All vectors are column vectors unless stated otherwise.

For a matrix A, we write its left-kernel as $\mathrm{LKer}(A) := \{\mathbf{x} \mid \mathbf{x}^{\intercal} \cdot A = \mathbf{0}\}$, which is a vector subspace of A's domain. The product of a vector space V with a matrix A is defined in the natural way as $\mathsf{V} \cdot A := \{\mathbf{x}^{\intercal} \cdot A \mid \mathbf{x} \in \mathsf{V}\}$.

We will need the following lemma about the extended gcd (greatest common divisor) algorithm in Section 4.

Lemma 1. *Consider any two integers $a, b \in \mathbb{Z}$. If a divides b, then the extended gcd algorithm outputs a as the greatest common divisor, along with the Bezout coefficients $(1, 0)$. Similarly, If b divides a, then the algorithm outputs b as the greatest common divisor, along with the Bezout coefficients $(0, 1)$. Recall that the Bezout coefficients are any integers α, β that satisfy the identity $\alpha a + \beta b = \gcd(a, b)$.*

2.2 Generic Group Model

In the generic group model, the group structure is hidden from an adversary. We use the definition of Maurer [27], since, in contrast to the model of Shoup [34], it doesn't allow for hash functions on group elements, since this would already result in short signatures, e.g. [32].

Specifically, the group is encapsulated in a black box, which has registers for group elements and only exposes them through labels to the outside. These labels are simply running register numbers and (unlike in Shoup's model) are not unique to a group element. (That is, several labels can reference the same group element.) A generic adversary can only interact with the group via a group operation oracle $\mathcal{O}_{\mathrm{grp}}$ and an equality test oracle $\mathcal{O}_{\mathrm{eq}}$. The group operation oracle takes two labels as input, internally computes the group operation on the group elements corresponding to the labels, writes the new group element to a new register and outputs the label of the new register. The equality test oracle takes two labels and outputs 1 iff the group elements corresponding to the labels are equal. For simplicity, we also add a multiplication oracle $\mathcal{O}_{\mathrm{mul}}$, which takes a label and an integer and returns a label to the group element multiplied by the integer, and assume that the GGM always outputs the same label for the same group element since this specific label can always be found by a generic adversary with polynomially many queries to the equality check oracle.

An adversary is called *generic*, if it works with only access to the generic group model. As the GGM is an information theoretic model, the running time of a generic adversary is typically measured by the number of its queries to the group oracles. It is called *pseudo-efficient*, if it makes polynomially many queries in the security parameter to its group oracles, yet its overall running time is (potentially) unbounded. If furthermore the overall running time is also polynomially bounded, then we call it *efficient*.

We will present a pseudo-efficient generic adversary in Section 3 and an efficient, standard model adversary in Section 4. Note that although a pseudo-efficient adversary in the generic group model doesn't immediately present an adversary on the schemes covered by our impossibility result, it is sufficient to rule out black-box constructions from generic groups alone. In other words, the result tells us that in order to make a signature scheme secure, we need another source of complexity, which we "factor out" through letting our adversary be unbounded outside of the generic group.

2.3 Signatures

We recall the standard definitions of syntax and security for digital signatures.

Definition 2 (Digital Signatures). *A digital signature scheme* $\mathsf{SIG} = \{\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}\}$ *consists of the following algorithms.*

- *The key generation algorithm* KeyGen *is probabilistic and on input of the security parameter* 1^λ *outputs a verification key and secret key* (vk, sk)*. We assume that* vk *implicitly defines the (finite) message space* $\mathcal{M}$*, which is superpolynomial in* λ*.*
- *The signing algorithm* Sign *takes a message* $m \in \mathcal{M}$ *and a secret key* sk *as input and returns a signature* σ*.*
- *The deterministic verification algorithm* Verify *takes a verification key, a message* $m \in \mathcal{M}$ *and a signature* σ *as input and returns 1 for accept and 0 for reject.*

We require that for all $(vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$ *and every message* $m \in \mathcal{M}$ *we have*

$$Pr[\mathsf{Verify}(vk, m, \mathsf{Sign}(sk, m)) = 1] = 1.$$

Definition 3 (UF-CMA Security). *We define the advantage of an adversary* $\mathcal{A}$ *against UF-CMA security (unforgeability against chosen message attack) of a signature scheme* SIG *as*

$$Adv^{\mathsf{UF\text{-}CMA}}_{\mathcal{A},\mathsf{SIG}}(\lambda) = Pr\left[\begin{array}{l|l} \mathsf{Verify}(vk, m^*, \sigma^*) = 1 & (vk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda) \\ \wedge\, m^* \notin \{m_1, \ldots, m_q\} & (m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(vk) \end{array}\right],$$

where $m_1, \ldots m_q$ *is the set of all messages queried to the signing oracle* $\mathsf{Sign}(sk, \cdot)$*.*

A weaker form of security is captured by unforgeability against random message attacks (UF-q-RMA security), where the adversary receives a set of q random messages and signatures.

Definition 4 (UF-q-RMA Security). *Let $q < |\mathcal{M}|$. We define the advantage of an adversary $\mathcal{A}$ against UF-q-RMA security (unforgeability against random message attack) of a signature scheme* SIG *as*

$$Adv^{\textsf{UF-}q\textsf{-RMA}}_{\mathcal{A},\textsf{SIG}}(\lambda) = Pr\left[\begin{array}{l|c} \textsf{Verify}(vk, m^*, \sigma^*) = 1 & (vk, sk) \xleftarrow{\$} \textsf{KeyGen}(1^\lambda) \\ \wedge\, m^* \notin \{m_1, \ldots, m_q\} & \forall i \in [q] : m_i \xleftarrow{\$} \mathcal{M} \setminus \{m_1, \ldots, m_{i-1}\} \\ & \forall i \in [q] : \sigma_i \xleftarrow{\$} \textsf{Sign}(sk, m_i) \\ & (m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}(vk, (m_i, \sigma_i)_{i \in [q]}) \end{array}\right].$$

Note that in UF-q-RMA security the messages are uniformly random without repetition, i.e., all messages are distinct. Moreover, we decided to make the parameter q explicit since it simplifies the exposition of our impossibility results. (In an alternative definition more closely related to UF-CMA security, the adversary would first specify the number of signatures q it would like to see. All our negative results also hold in this notion.)

3 Signature Schemes over Groups of Prime Order

In this section, we will show our impossibility result for signatures in the generic group model for prime order groups.

3.1 Algebraic Signatures

We now introduce our abstraction of algebraic signatures over a generic group $\mathbb{G}$. Intuitively, these signatures are limited by the fact that one doesn't have access to the representation of group elements. Specifically, in Mauer's GGM, one can't even define a consistent (hash) function on group elements not provided by the GGM itself, as the label of each group element depends on the order in which they are received and are therefore not consistent. This makes it hard to map group elements to group exponents consistently. Hence, in algebraic signatures all exponentiations in the verification have to be independent of the group elements of the signature and the verification key.

Definition 5. *An* algebraic signature scheme SIG *over $\mathcal{G} = (\mathbb{G}, p, P)$ of prime order p with parameters $n, k, \ell, \kappa \in \mathbb{N}$ polynomial in the security parameter λ is a digital signature scheme with the following structural properties.*

- *There exists efficiently computable functions $A : \{0,1\}^* \times \mathcal{M} \times \{0,1\}^\kappa \to \mathbb{Z}_p^{\ell \times n}$ and $B : \{0,1\}^* \times \mathcal{M} \times \{0,1\}^\kappa \to \mathbb{Z}_p^{\ell \times k}$. If s, m and t are clear, we write $A = A(s, m, t)$ and $B = B(s, m, t)$.*
- $\textsf{KeyGen}(1^\lambda)$ *outputs a keypair (vk, sk) with $sk \in \{0,1\}^*$ and*

$$vk = (\mathbf{X} = (X_1, \ldots, X_n)^\intercal, s) \in \mathbb{G}^n \times \{0,1\}^*.$$

- $\textsf{Sign}(sk, m)$ *outputs a signature*

$$\sigma = (\mathbf{Y} = (Y_1, \ldots, Y_k)^\intercal, t) \in \mathbb{G}^k \times \{0,1\}^\kappa.$$

– $\mathsf{Verify}(vk, m, \sigma)$ *returns* 1 *iff*

$$A(s, m, t) \cdot \mathbf{X} = B(s, m, t) \cdot \mathbf{Y}.$$

Recall that group $(\mathbb{G}, +)$ is written in additive form. Hence Verify checks whether ℓ group equations are fulfilled simultaneously.

3.2 Preparation

If B as in Definition 5 would be invertible for some m^*, t^*, then finding such a pair suffices to break the signature scheme. So we only consider signature schemes where this is not the case and use a different approach. As outlined in the introduction, our adversary will try to learn linear relations on the verification key elements and find a message for which a signature exists that verifies for all possible verification keys satisfying the linear relations already known. These relations form an affine space, but computing it requires group oracle queries. Since the number of group oracle queries an adversary is allowed to make is limited, we will not look at the relations directly but at the *dual space*, as it can be computed without group oracle access. Specifically, we will look for vectors $\mathbf{z} \in \mathbb{Z}_p^n$ s.t. $\mathbf{z}^\intercal \cdot \mathbf{X} = \mathbf{0}$. These can be found with the help of the following Lemma 6.

Lemma 6. *Let $A \in \mathbb{Z}_p^{\ell \times n}, B \in \mathbb{Z}_p^{\ell \times k}$ and $\mathbf{x} \in \mathbb{Z}_p^n$ for some prime p. Then the following statements are equivalent:*

$$\begin{aligned} \exists \mathbf{y} \in \mathbb{Z}_p^k: &\quad A \cdot \mathbf{x} = B \cdot \mathbf{y} && (1)\\ \forall \mathbf{z} \in \mathrm{LKer}(B): &\quad \mathbf{z}^\intercal \cdot A \cdot \mathbf{x} = 0 && (2)\\ \forall \mathbf{v} \in \mathrm{LKer}(B) \cdot A: &\quad \mathbf{v}^\intercal \cdot \mathbf{x} = 0. && (3) \end{aligned}$$

Notice that each signature will satisfy Eq. (1) and since A and B are $\mathbb{Z}_p$ matrices, the $\mathbf{v}$ in Eq. (3) can be computed without the group oracle. The specific usage of Lemma 6 will be described in Section 3.3.

Proof. We show the statement by proving circular implications in the order (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (1).

(1) $\Rightarrow$ (2): clear.

(2) $\Rightarrow$ (3): Let $\mathbf{v} \in \mathrm{LKer}(B) \cdot A$. Then there is a $\mathbf{z} \in \mathrm{LKer}(B)$, s.t. $\mathbf{v}^\intercal = \mathbf{z}^\intercal \cdot A$. Then we have $\mathbf{v}^\intercal \cdot \mathbf{x} = \mathbf{z}^\intercal \cdot A \cdot \mathbf{x} \overset{(2)}{=} 0$. Since $\mathbf{v}$ was chosen arbitrarily, this holds for all $\mathbf{v} \in \mathrm{LKer}(B) \cdot A$ and (3) follows.

(3) $\Rightarrow$ (1): Since (2) and (3) are clearly equivalent, we show (2) $\Rightarrow$ (1). Let $\forall \mathbf{z} \in \mathrm{LKer}(B) : \mathbf{z}^\intercal \cdot A \cdot \mathbf{x} = 0$. If $\mathbf{z}^\intercal \cdot A = 0$ for all $\mathbf{z}$, then $\mathrm{LKer}(A) \subseteq \mathrm{LKer}(B)$ and (1) follows, since then the range of A is a subspace of the range of B. So assume there is a $\mathbf{z} \in \mathrm{LKer}(B)$ s.t. $\mathbf{z} \cdot A \neq 0$. But then $\mathbf{z} \neq \mathbf{0}$ and $\mathbf{z}^\intercal \cdot (A \cdot \mathbf{x} - B \cdot \mathbf{y}) = 0$ for some $\mathbf{y} \in \mathbb{Z}_p^k$. But since $\mathbf{z} \notin \mathrm{LKer}(A)$ either $A \cdot \mathbf{x} = 0$ or $A \cdot \mathbf{x} = B \cdot \mathbf{y}$, which both imply (1).

3.3 Impossibility of Secure Algebraic Signatures

Theorem 7 (Impossibility of Algebraic Signatures with UF-q-RMA Security). *Let SIG be an algebraic signature scheme with parameters $n, k, \ell, \kappa \in \mathbb{N}$ over group $\mathbb{G}$ of prime order p. Then there exists a generic group adversary $\mathcal{A}$ with*

$$Adv^{\mathsf{UF}\text{-}n\text{-}\mathsf{RMA}}_{\mathcal{A},\mathsf{SIG}}(\lambda) \geq \frac{1}{n+1}$$

Specifically, $\mathcal{A}$ makes $Q_{\mathrm{mul}} = \ell(n+k)$ group multiplication queries to $\mathcal{O}_{\mathrm{mul}}$, $Q_{\mathrm{grp}} = \ell(n+k-2)$ group operation queries to $\mathcal{O}_{\mathrm{grp}}$, and additional $2^{\kappa} \cdot poly(n, k, \ell, \log(p))$ computation steps.

Note that adversary $\mathcal{A}$ is potentially pseudo-efficient as it makes polynomially many queries to its group oracles but its overall running time is exponential in κ.

We now provide an intuition for the proof. The central ingredient is Lemma 6, which is used in two ways during the proof. First, each signature is a valid solution to the verification equation system, so Eq. (3) holds and $\mathrm{LKer}(B(s, m, t)) \cdot A$ is exactly the space of linear relations on the verification key imposed by the signature. A and B output matrices over $\mathbb{Z}_p$, hence this space can be computed without GGM queries. On the other hand, if the adversary finds a message/bitstring pair (m^*, t^*) for which A and B satisfy Eq. (3), it knows that a solution to the verification equation exists for (m^*, t^*). Since $\mathbb{Z}_p$ is a field, the solution can be found with standard linear algebra techniques.

We will show that for a random message m^*, the probability that there exists a t^* s.t. (m^*, t^*) satisfies Eq. (3) is non-negligible. Intuitively, if we sample $n+1$ random messages and get signatures for n of them, we either learn the complete space of linear relations on the verification key or at least one of the messages doesn't introduce a new linear relation. In either case, at least one of the $n+1$ messages satisfies Eq. (3) and the adversary can forge a signature for that message.

Proof. We describe a generic attacker $\mathcal{A}$ with the properties stated above. First, $\mathcal{A}$ receives a verification key $vk = (\mathbf{X}, s)$ and $q := n$ message/signature pairs $(m_i, \sigma_i)_{i \in [n]}$, where $m_i \xleftarrow{\$} \mathcal{M} \backslash \{m_1, \ldots, m_{i-1}\}$ and $\sigma_i := (\mathbf{Y}_i, t_i) \xleftarrow{\$} \mathsf{Sign}(vk, m_i)$. Next, $\mathcal{A}$ computes

$$\mathsf{L} := \bigoplus_{j=1}^{n} \mathrm{LKer}(B(s, m_j, t_j)) \cdot A(s, m_j, t_j)$$

where $\bigoplus$ is the direct sum of the vector subspaces and $B(s, m_j, t_j)$ and $A(s, m_j, t_j)$ are the defining matrices from Definition 5. Note that L is a vector subspace of $\mathbb{Z}_p^n$. Then $\mathcal{A}$ chooses a random message $m^* \xleftarrow{\$} \mathcal{M} \setminus \{m_1, \ldots, m_n\}$ and for every $t \in \{0,1\}^{\kappa}$, $\mathcal{A}$ computes

$$\mathsf{K}(m^*, t) := \mathrm{LKer}(B(s, m^*, t)) \cdot A(s, m^*, t),$$

where s is the bit string from vk. $\mathcal{A}$ checks whether $\mathsf{K}(m^*, t) \subseteq \mathsf{L}$, i.e. if $\mathsf{K}(m^*, t)$ is a vector subspace of L. If $\mathcal{A}$ finds a pair (m^*, t^*) for which this condition holds, it continues. Otherwise, $\mathcal{A}$ aborts.

For the pair (m^*, t^*), $\mathcal{A}$ sets up the linear equation system

$$A(s, m^*, t^*) \cdot \mathbf{X} = B(s, m^*, t^*) \cdot \mathbf{Y}^*$$

and tries to solve it for $\mathbf{Y}^*$. If it finds a solution, it outputs its forgery $\sigma^* := (\mathbf{Y}^*, t^*)$ on message m^* and aborts otherwise.

We proceed to the analysis of $\mathcal{A}$.

Running time. First, note that $\mathcal{A}$ can check whether the condition $\mathsf{K}(m, t) \subseteq \mathsf{L}$ on the vector spaces holds without making any GGM queries and $\mathcal{A}$ can compute L without verifying the received signatures. Therefore $\mathcal{A}$ does not require any GGM queries before trying to solve the linear equation system, which takes at most $\ell(n + k)$ group multiplication queries and $\ell(n + k - 2)$ group operation queries, which yields the stated number of queries. Since $\mathcal{A}$ chooses a random message m^* and searches over all strings $t \in \{0, 1\}^\kappa$ to find a fitting t^* and computes the vector space $\mathsf{K}(m^*, t^*)$ for each potential t^*, its additional computation is bounded by $2^\kappa \cdot poly((n, k, \ell, \log(p))$, where the polynomial term consists mostly of subspace computations and membership tests. Therefore $\mathcal{A}$ is pseudo-efficient, unless $\kappa \in \mathcal{O}(\log(\lambda))$.

Correctness. To show correctness, we fix a verification key $vk = (\mathbf{X}, s)$ in the execution of $\mathcal{A}$.

For the moment, assume that $\mathcal{A}$ samples the distinct random messages $m_1, \ldots, m_{n+1}$ and chooses the i-th message as its forgery and queries the remaining n to the signing oracle. Define

$$\mathsf{L}_i := \bigoplus_{j \in [n+1] \setminus \{i\}} \mathsf{K}(m_j, t_j)$$

where t_j is some bitstring used in the signatures returned by the challenger for message m_j.

Then there exists at least one $i \in [n + 1]$ s.t. $\mathsf{K}(m_i, t_i) \subset L_i$ for some t_i.

Assume for contradiction, that no such i exists, i.e. $\mathsf{K}(m_i, t_i) \not\subset \mathsf{L}_i$ for all $i \in [n+1]$ and $t_i \in \{0, 1\}^\kappa$. But then the dimension of $L'_i = \bigoplus_{k \in [n+1] \setminus \{i,j\}} K(m_k, t_k)$ is smaller than the dimension of L_i for all $i \in [n + 1], j \in [n + 1] \setminus \{i\}$, since $K(m_j, t_j)$ increases the dimension of L_j and therefore is not included in any of the other $\mathsf{K}(m_i, t_i)$ and especially not in $\mathsf{K}(m_i, t_i)$. With the same argument, removing each $K(m_j, t_j)$ reduces the dimension of L_i by at least one. However since the dimension of L_i is at most $n - 1$ (as otherwise $\mathsf{L}_i = \mathbb{Z}_p^n$ and then $\mathsf{K}(m_i, t_i) \subseteq \mathsf{L}_i$), removing n messages would reduce its dimension to -1, which is a contradiction.

So by choosing a random message from a set of $n + 1$ messages, the adversary chooses a message m^* which satisfies Eq. (3) together with some t^* with probability at least $\frac{1}{n+1}$. But since all messages are random, this is the same as saying that the probability of the last message, i.e. m_{n+1} being this message is at least $\frac{1}{n+1}$ and this also holds if the first n messages are randomly chosen by the challenger and only the last message is chosen by the attacker.

Therefore $\mathcal{A}$ finds a pair (m^*, t^*) such that $A(s, m^*, t^*)$ and $B(s, m^*, t^*)$ satisfy Equation (3) from Lemma 6 for the verification key vector $\mathbf{X}$ with probability at least $\frac{1}{n+1}$. This implies that the verification equation system has a solution and it can be found using standard linear algebra techniques since $\mathbb{Z}_p$ is a field. Since a signature is valid, iff it satisfies the verification equation this solution is exactly a valid signature and $\mathcal{A}$ wins the UF-n-RMA game.

4 Signature Schemes over Groups of Unknown Order

In this section, we describe a linear attack concerning a specific form of signatures over groups of unknown order. This attack has implications in particular on factoring- and RSA-based signatures. We start first by defining a particular type of signatures that we can attack. We call this type of signatures simplified algebraic signatures.

4.1 Simplified Algebraic Signatures

Unlike in the previous case where the group order was known, here the signatures rely on the algebraic structure of a group of (potentially) unknown order, such as in RSA signatures.

Compared to our formalization of algebraic signatures from Definition 5, the verification key of *simplified* algebraic signatures does not contain the s element anymore. This change is without loss of generality: namely, since the results in this section can be formulated in the standard model (without generic groups), linear coefficients can depend on the full representation of group elements in the verification key. Any additional information (that was previously contained in s) can now be encoded with group elements in vk.

Furthermore, signatures in simplified algebraic signature schemes do not contain the string t anymore. This is in fact a restriction, and it is caused by the fact that our attacks from this section need to be efficient. Hence, we cannot afford to run a brute-force search over a suitable t during an attack here (unlike in the attack from Section 3).

Definition 8. *Let λ denote the security parameter and $\mathcal{G} = (\mathbb{G}, N, P)$ be a group of order $N \in \mathbb{N}$ (that may or may not be known, with N having $O(\lambda)$ bits). A* simplified algebraic signature scheme SIG *over $\mathcal{G}$ with parameters $n, k, \ell \in \mathbb{N}$ (polynomial in λ) is a digital signature scheme with the following structural properties.*

- *There exist efficiently computable functions $A : \{0,1\}^* \times \mathcal{M} \to \mathbb{Z}_N^{\ell \times n}$ and $B : \{0,1\}^* \to \mathbb{Z}_N^{\ell \times k}$. If m is clear, we write $A := A(vk, m)$ and $B := B(vk)$ respectively.*
- $\mathsf{KeyGen}(1^\lambda)$ *outputs a keypair (vk, sk) with $sk \in \{0,1\}^*$ and*

$$vk = (\mathbf{X} = (X_1, \dots, X_n)^\top) \in \mathbb{G}^n.$$

– $\mathsf{Sign}(sk, m)$ *outputs a signature*

$$\sigma = (\mathbf{Y} = (Y_1, \ldots, Y_k)^\top) \in \mathbb{G}^k.$$

– $\mathsf{Verify}(vk, m, \sigma)$ *returns* 1 *iff*

$$A(vk, m) \cdot \mathbf{X} = B(vk) \cdot \mathbf{Y}.$$

4.2 Hermite Normal Form

We present one of the main technical tools we are going to use in this section and which allows us to utilize linear algebra over the ring of integers.

Definition 9 (Hermite Normal Form [3,29]**).** *An $n \times m$ matrix H over $\mathbb{Z}$ is in* Hermite Normal Form (HNF) *if $H = \mathbf{0}$ or $H \neq \mathbf{0}$ and there exists an integer r with $1 \leq r \leq \min(n, m)$, such that:*

– *the first r columns are non-zero, i.e., $\mathbf{h}_{*j} \neq \mathbf{0}$ for all $j \in [r]$.*
– *there is a sequence of integers $1 \leq n_1 < n_2 < \ldots < n_r \leq n$, such that:*
  - *$h_{i,j} = 0$ for $j \in [m], i < n_j$. Also, $h_{i,j} = 0$ for $j > r$ and any $i \in [n]$.*
  - *$h_{n_j,j} > 0$, for all $1 \leq j \leq r$. Moreover, all entries of H on rows n_j for $j \in [r]$ are non-negative and $h_{n_j,j}$ is strictly greater than all other elements on the n_j^{th} row, namely: $0 \leq h_{n_j,k} < h_{n_j,j}$ for all $j \in [r], k \in [j-1]$.*

Note that r from Definition 9 coincides with the column-rank of H. Matrix H will therefore be in HNF if it has the following shape, where $\mathbf{0}$ denotes the all-zero matrix in $\mathbb{Z}^{n,m-r}$ and $*$ stands for any element of $\mathbb{Z}$:

$$\left(\begin{array}{cccc|c}
0 & 0 & & 0 & \\
\vdots & \vdots & & & \\
0 & 0 & & & \\
h_{n_1,1} & 0 & & \vdots & \\
* & \vdots & & & \\
* & 0 & & 0 & \\
* & h_{n_2,2} & & \vdots & \mathbf{0} \\
* & * & \ddots & 0 & \\
* & * & \ddots & h_{n_r,r} & \\
* & * & \ddots & \vdots & \\
* & * & \ddots & * &
\end{array}\right)$$

The top-most non-zero element $h_{n_j,j}$, $j \in [r]$ on each column is called a *pivot*. The second condition in Definition 9 tells us that all elements on a row with a pivot are non-negative and must be strictly smaller than the pivot. No condition is enforced on elements that are on a row without a pivot, meaning they might be negative.

Note that we use this (more-general) definition (see for example [3]) because we need to accommodate matrices which are not-necessarily square, and for which the column-rank is not necessarily maximal.

Lemma 10 (Existence and uniqueness of HNF [33]**).** *For any matrix $M \in \mathbb{Z}^{n \times m}$, there exists a unique matrix $\mathsf{HNF}(M) \in \mathbb{Z}^{n \times m}$ in Hermite Normal Form such that:*

$$\mathsf{ColumnSpace}(\mathsf{HNF}(M)) = \mathsf{ColumnSpace}(M).$$

Lemma 11 (A polynomial-time algorithm for HNF [13]**).** *For any $n \times m$ matrix M, computing its Hermite Normal Form can be realized in polynomial time.*

The following lemma is well-known:

Lemma 12. *Let $A \in \mathbb{Z}^{n \times m}, H = \mathsf{HNF}(A)$ and $\mathbf{c} \in \mathbb{Z}^n$. If $\mathbf{c} \in \mathsf{ColumnSpace}(H)$ then we can find in polynomial time (in the bitlength of its input) integer vectors $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ such that $\mathbf{c} = H \cdot \boldsymbol{\beta}$ and $\mathbf{c} = A \cdot \boldsymbol{\alpha}$.*

4.3 An Inefficient AddColumn Procedure for matrices in HNF

In this section we describe a straightforward inefficient algorithm which on input matrix H in Hermite Normal Form and column vector $\mathbf{d}$, will compute the HNF of $[H \mid \mathbf{d}]$. While inefficient, this algorithm will simplify our proof by allowing us to analyze the impact of successively adding columns to a matrix H on its HNF. Since the HNF is unique, the reasoning we come to using the inefficient algorithm will extend to the efficient one, since we are only concerned with what happens to the HNF, and not its intermediate results.

The AddColumn Algorithm

Input: $n \times m$ matrix $H = [B \mid \mathbf{0}]$ in Hermite Normal Form and column vector $\mathbf{d}$. Assume without loss of generality that $H = B$.
Output: $\mathsf{HNF}([H \mid \mathbf{d}])$.

The algorithm iterates over the non-zero columns of H as follows.

1. **Initialization**: The algorithm initializes $E(0) = B$ and $\mathbf{c}(0) = \mathbf{d}$. At step i, matrix $E(i)$ is initialized as $E(i) \leftarrow E(i-1)$ and vector $\mathbf{c}(i) \leftarrow \mathbf{c}(i-1)$. Additionally, at step i, vector $\mathbf{c}(i)$ has its first $i-1$ elements set to 0. Values $r(i), n_1(i), \ldots, n_r(i)$ correspond to the $r, n_1 \ldots n_r$ values of $E(i)$ from Definition 9. Since these indices may change, they depend on the iteration number i. If $c(i)_i$ is 0, then we skip iteration i and move on to step $i+1$.

Otherwise, if $c(i)_i \neq 0$, we let s be the smallest index such that $n_s(i-1) \geq i$. We write:

$$E(i) = \begin{pmatrix} 0 & & & \mathbf{0} \\ \vdots & & & \mathbf{0} \\ 0 & & & \mathbf{0} \\ e_{n_1(i),1}(i) & & & \mathbf{0} \\ e_{n_1(i)+1,1}(i) & \ddots & & \\ \vdots & & e_{n_s(i),s}(i) & \\ & & \vdots & F(i) \\ e_{n,1}(i) & & e_{n,s}(i) & \end{pmatrix}$$

In the right lower corner of $E(i)$ is matrix $F(i) \in \mathbb{Z}^{(n-n_s(i))\times(m-s)}$, which does not change during iteration i. All entries $e(i)_{k,j}$ with $k < n_j(i)$ are 0, for all $j \in [r]$ and $k \in [n_j(i)-1]$. We also have $e(i)_{k,j} = 0$, for all $k \in [n]$ and $j > r(i)$.

2. **Column Index Selection**: Recall that s is the smallest index such that $n_s(i-1) \geq i$. What this means is that the s^{th} column is the first column of $E(i)$ that can be modified using element $c(i)_i$. Now, we distinguish the following cases:
   (a) **Column Insertion Step**: If $n_s(i-1) > i$, we insert vector $\mathbf{c}(i)$ before the s^{th} column of $E(i)$. Namely, $\mathbf{c}(i)$ becomes the s^{th} column of $E(i)$ and matrix $E(i)$ increases its column dimension by 1. We then set $\mathbf{c}(i)$ to the all-zero vector and after the modular reduction phase (which we will soon describe as well), we will output $E(i)$ as the HNF of $[H|\mathbf{d}]$.
   (b) **GCD step**: Otherwise, we have $n_s(i-1) = i$. We then know that $e(i)_{i,s} \neq 0$, since $e(i)_{i,s}$ is the top-most non-zero element on the s^{th} column (the pivot). We compute $g \leftarrow \gcd(e(i)_{i,s}, c(i)_i) = \alpha e(i)_{i,s} + \beta c(i)_i$, where $\alpha, \beta \in \mathbb{Z}$ are the Bezout coefficients computed by the extended greatest common divisor algorithm. We aim to replace $e_{i,s}(i)$ with $g = \gcd(e(i)_{i,s}, c(i)_i)$, while preserving the column space of $E(i)$:
      - The s^{th} column of $E(i)$ is modified as $\mathbf{e}(i)_{*s} \leftarrow \alpha\mathbf{e}(i)_{*s} + \beta\mathbf{c}(i)$. Note that the first $(i-1)$ entries of $c(i)$ are 0, which means that $e(i)_{i,s}$ remains the top-most non-zero element on the s^{th} column (the pivot), as required by the HNF condition (Definition 9).
      - Vector $\mathbf{c}(i) \leftarrow \mathbf{c}(i) - (c(i)_i/g)\cdot\mathbf{e}(i)_{*s}$. This replaces the i^{th} component of $\mathbf{c}(i)$ with 0.
3. **Modular Reduction Phase**: Finally, the algorithm has to ensure that $0 \leq e(i)_{n_j,k} < e(i)_{n_j,j}$ for all $k \in [m], j \in [r]$. This is done by reducing the large entries modulo the pivots (the top-most non-zero elements on each column). To preserve the column space, the algorithm uses the following procedure:

```
for $j = 1$ to $m$ do
  for $k = j + 1$ to $r$ do
    if $e(i)_{n_k(i),j} \geq e(i)_{n_k(i),k}$ then
```

```
      $\mathbf{e}(i)_{*j} \leftarrow \mathbf{e}(i)_{*j} - \lfloor \frac{e(i)_{n_k(i),j}}{e(i)_{n_k(i),k}} \rfloor \mathbf{e}(i)_{*k}$
    else if $e(i)_{n_k(i),j} < 0$ then
      $\mathbf{e}(i)_{*j} \leftarrow \mathbf{e}(i)_{*j} + \lceil \frac{e(i)_{n_k(i),j}}{e(i)_{n_k(i),k}} \rceil \mathbf{e}(i)_{*k}$
    end if
  end for
end for
```

At this stage $E(i)$ is in HNF and $\mathbf{c}(i)$ has its first i entries equal to 0. The algorithm is now ready to move on to the next i, making the same computations for $E(i+1)$. Note that all operations of AddColumn are expressed as operations on the columns of the matrix $E(i)$. This means that $E(i)$ has the same column space as $E(i-1)$.

4. **Output**: If the algorithm made no column insertions, then the output is $[E(n)\|\mathbf{0}]$, where here $\mathbf{0} \in \mathbb{Z}^{n\times 1}$. If at some point the algorithm made a column insertion, the output will be $E(n)$.

Remark on the running time of AddColumn. The algorithm described above is (potentially) inefficient because the intermediate values computed by the algorithm are not shown to be bounded. A polynomial time algorithm for computing $\mathsf{HNF}([H \mid \mathbf{b}])$ can be found in [13,29].

Lemma 13 (Decreasing pivots). *Consider $n, m \in \mathbb{N}^*$. Let $H \in \mathbb{Z}^{n\times m}$ be a matrix in* HNF *and of column-rank r. Consider a column vector $\mathbf{b}$ and let $H' := \mathsf{AddColumn}(H, \mathbf{b}) = \mathsf{HNF}([H \mid \mathbf{b}])$. Let $r, n_1, \ldots, n_r$ be the column-rank and pivot indices of H, and $r', n'_1, \ldots, n'_r$ the values corresponding to H' (as in Definition 9).*

Then we must have one of the following cases:

1. *$r' > r$, i.e. the column rank of H' is strictly greater than the one of H.*
2. *$r' = r$ and then the positions of the pivots remain the same ($n_j = n'_j$, for all $j \in [r]$). Moreover, all pivot entries of H' are smaller than or equal to the corresponding pivot elements of H, i.e. $h'_{n_j,j} \leq h_{n_j,j}$, for all $j \in [n]$. Furthermore, at least a pivot entry of H' is smaller by at least a factor of 2 than the corresponding pivot of H, i.e. $h'_{n_j,j} \leq h_{n_j,j}/2$, for some $j \in [r]$.*
3. *$\mathbf{b} \in \mathsf{ColumnSpace}(H)$.*

Proof. We consider the following statements:

– $r' < r$ **cannot hold.** Let $E(i)$ and $\mathbf{c}(i)$ be the intermediary values computed by AddColumn. We show first that $r' < r$ cannot hold. Observe that the algorithm may insert one column (and only one) during its execution, which introduces a new pivot element. Apart from that, each pivot element in H' is computed as the greatest common divisor of a non-zero pivot element in $E(i)$ and a non-zero element of $\mathbf{c}(i)$. Moreover, the modular reduction phase does not modify the value of the pivots. Therefore, pivots in H cannot be set to 0 and thus $r \leq r'$.

- **Case 1 does not hold implies pivot positions are the same.** If $r = r'$, this implies that the algorithm never inserts the vector $\mathbf{c}(i)$ as a column of $E(i)$. As explained in the previous paragraph, pivots in H' are computed as the greatest common divisor of a non-zero pivot and $E(i)$ and a non-zero element of $\mathbf{c}(i)$. Additionally, since pivots in $E(i)$ cannot become 0 and no column insertion is made, the pivot positions also remain unchanged.
- **Case 1 does not hold implies smaller pivots.** By contradiction, assume for now that the pivot entries $h'_{n_j,j}$ are not smaller or equal than $h_{n_j,j}$, for all $j \in [r]$. Since we are in the case of no column insertions, observe that $e(j)_{n_i,i}$ can only change its value in iteration n_i of $\mathsf{AddColumn}$. This means that $h'_{n_i,i}$ is equal to the value $e(i)_{n_i,i}$ has at the end of iteration n_i. Then, since $h_{n_i,i} \neq 0$, $e(i)_{n_i,i} = \gcd(h_{n_i,i}, c(i)_{n_i})$, therefore $e(i)_{n_i,i}$ divides $h_{n_i,i}$. Thus, we have $e(i)_{n_i,i} \leq h_{n_i,i}$, which implies $h'_{n_i,i} \leq h_{n_i,i}$.
- **Case 1 does not hold and equal pivots imply Case 3.** Now assume by contradiction that $h'_{n_j,j} = h_{n_j,j}$, for all $j \in [r]$. This means that $e(i)_{n_j,j}$ remain the same throughout the execution of the $\mathsf{AddColumn}$ algorithm. From the description of $\mathsf{AddColumn}$, we have that $e(i)_{n_j,j} = \gcd(e(i-1)_{n_j,j}, c(i)_{n_j})$. We use Lemma 1 and notice that, if the pivot elements $e(i)_{n_j,j}$ never change, then neither can the non-zero elements which are not pivots. That is because they already satisfy $0 \leq e_{n_k,j} < e_{n_k,k}$, for all $k \in [r], j \in [k-1]$ at the start of the algorithm, so the modular reduction phase cannot further reduce these elements.
  Recall that we defined $s(i)$ as the smallest index such that $n_s(i-1) \geq i$. Since by hypothesis, case 1 does not hold, this implies that $n_s(i-1) = i$. But then, it must hold that, at the beginning of iteration i, $e(i)_{n_{s(i)},s(i)}$ was already equal to $\gcd(e(i)_{n_{s(i)},s(i)}, c(i)_{n_{s(i)}})$. Therefore, $c(i)_{n_{s(i)}}$ is already divisible by $e(i)_{n_{s(i)},s(i)}$ at the beginning of iteration i. Since this holds for all iterations $i \in [n]$, this implies that $\mathbf{b}$ can be expressed as a linear combination of the columns of H, therefore $\mathbf{b} \in \mathsf{ColumnSpace}(H)$.
- **Case 1 and Case 3 do not hold imply Case 2.** Let's now assume that $\mathbf{b} \notin \mathsf{ColumnSpace}(H)$. From the argument above, we know that for some iteration $i \in [n]$ and index $s(i) \in [r]$, at least one pivot element $e(i)_{n_{s(i)},s(i)} < e(i-1)_{n_s(i),s(i)}$. But $e(i)_{n_{s(i)},s(i)} = \gcd(e(i-1)_{n_s(i),s(i)}, c(i)_i)$, so $e(i)_{n_{s(i)},s(i)}$ must be smaller by at least a factor of 2 than $e(i-1)_{n_{s(i)},s(i)}$. Since $e(i)_{n_{s(i)},s(i)}$ is not changed in iterations $(i+1)\ldots n$, this implies that $h'_{n_{s(i)},s(i)} \leq h_{n_{s(i)},s(i)}/2$.

This concludes the proof of Lemma 13.

4.4 Impossibility of Simplified Algebraic Signatures

Theorem 14 (Impossibility of Simplified Algebraic Signatures with UF-CMA Security). *Let* SIG *be a simplified algebraic signature scheme with parameters* $\lambda, n, k, \ell \in \mathbb{N}$ *over group* $\mathbb{G}$ *of (possibly) unknown order* N *and message space superpolynomial in* λ*. Then there exists a PPT adversary* $\mathcal{A}$ *with*

$$Adv^{\mathsf{UF\text{-}CMA}}_{\mathcal{A},\mathsf{SIG}}(\lambda) = 1$$

The adversary makes at most $Q_S = (n\ell)^2 \cdot |\tau_{\max}|$ signature queries, where $|\tau_{max}|$ is the bitlength of the largest entry in the matrices $A(vk, m)$ for the messages that will be queried (we will argue that $|\tau_{\max}| = \text{poly}(n\log(N) + |m|)$). In addition, $|\mathcal{M}|$ does not have to be superpolynomial, it suffices to require $|\mathcal{M}| \geq Q_S + 1$.

Letting T_{linear} denote an upper bound on the run-time of all invocations of the polynomial-time HNF algorithm and $T_{\max}$ be an upper-bound on the polynomial running time of functions A, our attack runs in $O(Q_S \cdot (T_{\max} + T_{\text{linear}})) = O\Big((n\ell)^2 \cdot |\tau_{\max}| \cdot (T_{\max} + T_{\text{linear}})\Big)$.

Proof. We describe our PPT adversary $\mathcal{A}$. Recall that $A(vk, m)$ is efficiently computable, this means that there exists a value τ_{max} whose bitlength is greater or equal than all the entries of the matrices $A(vk, m)$ that will be queried. Since the input length of the functions computing $A(vk, m)$ is $n\log(N) + |m|$, we have that the bitlength $|\tau_{\max}|$ is polynomially small with $|\tau_{max}| = \text{poly}(n\log(N) + |m|)$.

1. **Setup Phase**. The challenger runs KeyGen, generating sk and $vk = \mathbf{X}$ and sends vk to $\mathcal{A}$.
2. **Discovery Phase**. Let $\mathbf{a}_{i*}(vk, m)^\top$ denote the i^{th} row of $A(vk, m)$ and $\mathbf{b}_{j*}(vk)^\top$ be the j^{th} row of $B(vk)$. The adversary initializes matrix $H(0)$ to be the empty matrix. At iteration i, the adversary picks a uniformly random message $m_i \xleftarrow{\$} \mathcal{M} \setminus \{m_1, \ldots, m_{i-1}\}$. It then computes the column vector $\mathbf{c}(i) = [\mathbf{a}_{1*}(vk, m_i)^\top | \ldots | \mathbf{a}_{\ell*}(vk, m_i)^\top]^\top \in \mathbb{Z}^{n\ell}$ and builds matrix $D(i) = [H(i-1)|\mathbf{c}(i)] \in \mathbb{Z}^{n\ell \times i}$. Let $H(i) = \mathsf{HNF}(D(i))$, where $\mathsf{HNF}(D(i))$ denotes the (column-style) HNF of matrix $D(i)$. The adversary checks whether $H(i) = [H(i-1)|\mathbf{0}]$, if that is the case, it means that $\mathbf{c}(i) \in \mathsf{ColumnSpace}(H(i-1))$ (i.e. $\mathbf{c}(i)$ is in the linear span of the columns of $H(i-1)$). Let $\mathbf{h}(i-1)_{*j}$ denote the j^{th} column of $H(i-1)$. Using linear algebra over $\mathbb{Z}$ (see for example [33]), the adversary can efficiently compute a linear combination of the columns of $H(i-1)$. More specifically, it can find a vector $\boldsymbol{\beta} \in \mathbb{Z}^i$ such that $\mathbf{c}(i) = \sum_j \beta_j \mathbf{h}(i-1)_{*j}$ through a matrix-vector multiplication (whose complexity can be bounded by T_{linear}). From Lemma 12, this allows us to also recover a vector $\boldsymbol{\alpha} \in \mathbb{Z}^i$ such that $\mathbf{c}(i) = \sum_j \alpha_j \mathbf{c}(j)$, where $\mathbf{c}(i)$ was defined as $\mathbf{c}(i) = [\mathbf{a}_{1*}(vk, m_i)^\top | \ldots | \mathbf{a}_{\ell*}(vk, m_i)^\top]^\top \in \mathbb{Z}^{n\ell}$.
3. **Signing Queries**: Consider $j \in [i-1]$, then for all $\alpha_j \neq 0$, the adversary makes a signing query on m_j and receives signature $\mathbf{Y}_{m_j} = (Y_{m_j,1} \ldots Y_{m_j,k})$.
4. **Forgery Phase**: Consider the following matrix of group elements:

$$\mathbf{W} = \begin{pmatrix} \mathbf{Y}_{m_1}^\top \\ \hline \vdots \\ \hline \mathbf{Y}_{m_{i-1}}^\top \end{pmatrix} \in \mathbb{G}^{(i-1) \times k}$$

   At this stage, the adversary can compute a forged signature for message $m^* = m_i$ as $(\mathbf{Y}^*)^\top = \boldsymbol{\alpha}^\top \mathbf{W}$ and it outputs the forgery $\mathbf{Y}^*$.

Correctness of the attack. Let's assume for now that the adversary has output a forgery. Since $\mathbf{c}(i) = \sum_j \alpha_j \mathbf{c}(j)$, where $\mathbf{c}(j)$ was defined as $\mathbf{c}(i) = [\mathbf{a}_{1*}(vk, m_i)^\top | \ldots | \mathbf{a}_{\ell*}(vk, m_i)^\top]^\top$, we have the following intermediary result:

$$\sum_{j=1}^{i-1} \Big(\alpha_j \cdot A(vk, m_j) \Big) = A(vk, m^*). \tag{4}$$

The forgery is an accepting signature, because:

$$\begin{aligned} B(vk)\mathbf{Y}^* =& B(vk)\mathbf{W}^\top \boldsymbol{\alpha} \\ =& \big(B(vk)\mathbf{Y}_{m_1} \big| \ldots \big| B(vk)\mathbf{Y}_{m_{i-1}} \big) \cdot \boldsymbol{\alpha} \\ \overset{(*)}{=}& \big(A(vk, m_1)\mathbf{X} \big| \ldots \big| A(vk, m_{i-1})\mathbf{X} \big) \cdot \boldsymbol{\alpha} \\ =& \sum_{j=1}^{i-1} \Big(\alpha_j \cdot A(vk, m_j)\mathbf{X} \Big) = \Big(\sum_{j=1}^{i-1} \alpha_j \cdot A(vk, m_j) \Big) \mathbf{X} \\ =& A(vk, m^*)\mathbf{X}. \end{aligned}$$

Equality (*) holds because signatures $\mathbf{Y}_{m_j}$ are correct for all $j \in [i-1]$. We have shown that $A(vk, m)\mathbf{X} = B(vk)\mathbf{Y}^*$, which implies that the forgery is a valid signature. The last equation uses Equation (4).

Running Time. The adversary can only compute a forgery if for some iteration i, it holds that $\mathbf{c}(i) \in \mathsf{ColumnSpace}(H(i-1))$. We show that this must occur after a polynomial number of iterations. From Lemma 13, we have that at least one of the following cases must hold:

1. The column rank of $H(i)$ is strictly larger than the rank of $H(i-1)$.
2. The rank is unchanged, so are the positions of the pivots, but at least one pivot of $H(i)$ decreases by at least a factor of 2. Furthermore, all other pivots are smaller than or equal to the previous ones, i.e. $h(i)_{n_j,j} \le h(i-1)_{n_j,j}$, for all $j \in [n]$.
3. Or $\mathbf{c}(i) \in \mathsf{ColumnSpace}(H(i-1))$.

Note that even though Lemma 13 is proven by reasoning about the inefficient algorithm AddColumn, since the HNF is unique, the proof also applies to poly-time HNF algorithms (Lemma 11). Therefore, at each iteration, we have the following possibilities:

1. The rank of $H(i)$ increases. This can only happen at most $n\ell$ times.
2. Column-rank and pivot positions are unchanged, but at least one pivot decreases by a factor of 2, while no other pivot increases. Since the HNF is applied on matrices $A(vk, m)$ with each entry bounded by $\tau_{\max}$, this can happen at most $|\tau_{\max}| \cdot n\ell$ times (we can have at most $n\ell$ pivots). The pivots become smaller by one bit, until they become 1, and do not decrease further.
3. It holds that $\mathbf{c}(i) \in \mathsf{ColumnSpace}(H(i-1))$, and we can forge a signature.

Therefore, this means that after at most $(n\ell)^2 \cdot |\tau_{\max}|$ iterations, we will end up in the third case, when $H(i) = [H(i-1)|\mathbf{0}]$ and we can compute a forgery. Since computing the HNF is a polynomial-time algorithm and the number of iterations is polynomial, this means that the adversary also runs in polynomial time.

Theorem 15 (Impossibility of Simplified Algebraic Signatures with UF-q-RMA Security). *Let* SIG *be a simplified algebraic signature scheme with parameters $n, k, \ell \in \mathbb{N}$ over group $\mathbb{G}$ of (possibly) unknown order N, and let Q_S be defined as in Theorem 14. There exists a PPT adversary $\mathcal{A}$ with*

$$Adv^{\mathsf{UF}\text{-}Q_S\text{-}\mathsf{RMA}}_{\mathcal{A},\mathsf{SIG}}(\lambda) = \frac{1}{Q_S + 1}$$

Proof. Using Theorem 14, we know that there exists an adaptive adversary $\mathcal{A}'$ making Q_S signature queries, which wins the adaptive UF-CMA game with probability 1. The adversary $\mathcal{A}'$ picks Q_S different messages m_i uniformly at random and requests signatures once it finds a linear dependency between the columns of $A(vk, m_i)$ and $\{A(vk, m_j)\}$, for $j \in [i-1]$.

Adversary $\mathcal{A}'$ can be modified to an adversary $\mathcal{A}''$ which does not choose the messages m_i itself. Adversary $\mathcal{A}''$ is guaranteed to adaptively receive Q_S uniformly random, distinct messages m_i sampled by the challenger of a hybrid game Hyb. In Hyb, the challenger samples the Q_S messages uniformly at random, but it sends them to the adversary $\mathcal{A}''$ one by one, at each stage the adversary sees message m_j and can adaptively decide whether it also wants to query the signing oracle on m_j.

Upon receiving each m_i, $\mathcal{A}''$ computes $\mathbf{c}(i) = [\mathbf{a}_{1*}(vk, m_i)^\top | \dots | \mathbf{a}_{\ell *}(vk, m_i)^\top]^\top \in \mathbb{Z}^{n\ell}$.

- If $\mathbf{c}(i) \in \mathsf{ColumnSpace}(H(i-1))$, then $\mathcal{A}''$ does not ask for a signature and forges in the same way as $\mathcal{A}'$.
- Otherwise, $\mathcal{A}''$ cannot forge yet and therefore asks for a signature for message m_i.

Is $\mathcal{A}''$ does not manage to forge for any of the Q_S messages it has received, then it samples one more message from $\mathcal{M} \setminus \{m_1, \dots, m_{Q_S}\}$ itself and from Theorem 15, it will be able to forge on this final message. Since $\mathcal{A}'$ succeeds, this implies that $\mathcal{A}''$ successfully computes a forgery in Hyb as well. Now, we construct the PPT adversary $\mathcal{A}$ against the UF-Q_S-RMA-game. This adversary will simply run $\mathcal{A}''$, simulating its view. In order to do that successfully, $\mathcal{A}'$ must initially guess the index i on which $\mathcal{A}''$ will compute a forgery. It follows that $\mathcal{A}$ succeeds with probability $\frac{1}{Q_S+1}$.

Remark 16. We can also consider a slight generalization of Definition 8 to account for the additional element t from Definition 5. Specifically, we could allow verification to check equations of the form:

$$A(vk, m, t) \cdot \mathbf{X} = B(vk) \cdot \mathbf{Y}.$$

The value $t \in \{0,1\}^\kappa$ is then part of the signature. Note that unlike in Definition 5, matrices B are not allowed to depend on t here. Our attack can be generalized to this setting and remains in polynomial-time, as long as $\kappa = O(\log(\lambda))$.

5 Extension: BLS signatures instantiated with algebraic hash functions are insecure

In this section, we show that BLS signatures [8] (which are formulated in the random oracle model) cannot be implemented with an "algebraic" standard-model hash function (such as a programmable hash function [19]).

Definition 17 (Algebraic hash function). *An algebraic hash function over a group $\mathbb{G}$ and with message space $\mathcal{M}$ consists of two PPT algorithms:*

- *A key generation algorithm* HGen *that outputs an evaluation key hk. We assume that hk specifies a vector $\mathbf{X} = (X_1, \ldots, X_n)^\intercal \in \mathbb{G}^n$ of group elements.*
- *An evaluation algorithm* Eval *that, on input hk and $m \in \mathcal{M}$, outputs a hash value*
$$\mathsf{H}_{hk}(m) = A(hk, m)^\intercal \cdot \mathbf{X} \in \mathbb{G}$$
for a public and efficiently computable function A with output in $\mathbb{Z}^n$.

In a nutshell, algebraic hash functions construct their output through generic group operations from a sequence $\mathbf{X}$ of public group elements (defined in the hashing key *hk*). Popular constructions of programmable hash functions (e.g., [35,19,18] are algebraic hash functions.

In this section, we want to show that the attack in Theorem 14 can be adjusted to also work against another type of signatures, which we refer to as plain algebraic signatures in pairing groups. This class of signatures generalizes the BLS signature when the BLS hash function is modelled as an algebraic hash function. What is different from Definition 8, is that Definition 18 supports verification equations which apply a pairing operation on certain elements of the verification key along with other parts of the verification key. In particular, this means that the signature can consist of group elements whose implicit exponents correspond to quadratic relations in the implicit exponents of the group elements in the verification key.

Definition 18 (Plain Algebraic Signatures in Pairing Groups). *Let λ denote the security parameter and $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, P, e)$ be a symmetric pairing group of order $N \in \mathbb{N}$ (that may or may not be known, with N having $O(\lambda)$ bits). Consider also* $(\mathsf{HGen}, \mathsf{Eval})$ *to be an algebraic hash function. A* plain algebraic signature scheme SIG *over $\mathcal{G}$ with parameters $k, \gamma \in \mathbb{N}$ (polynomial in λ) is a digital signature scheme with the following structural properties.*

- *There exist efficiently computable functions $A : \{0,1\}^* \times \mathcal{M} \to \mathbb{Z}_N^{\gamma \times 1}$ and $B : \{0,1\}^* \to \mathbb{Z}_N^{1 \times k}$. If m is clear, we write $A := A(hk, m)$ and $B := B(vk)$ respectively.*

- $\mathsf{KeyGen}(1^\lambda)$ *outputs a keypair (vk, sk) with* $sk \in \{0,1\}^*$ *and*

$$vk = (X, \mathbf{X}^{hk}) = (X, (X_1^{hk}, \ldots, X_\gamma^{hk})^\top) \in \mathbb{G} \times \mathbb{G}^\gamma,$$

  where $hk := \mathbf{X}^{hk}$ *is the hash key of the algebraic hash function, generated using the* HGen *algorithm.*
- $\mathsf{Sign}(sk, m)$ *outputs a signature* σ*, with:*

$$\sigma = (\mathbf{Y} = (Y_1, \ldots, Y_k)^\top) \in \mathbb{G}^k.$$

- $\mathsf{Verify}(vk, m, \sigma)$ *returns* 1 *iff*

$$e(\mathsf{H}_{hk}(m), X) = e(A(hk, m)^\intercal \cdot \mathbf{X}^{hk}, X) = e(P, B(vk) \cdot \mathbf{Y}).$$

The result in Theorem 14 extends to the signatures in Definition 18:

Theorem 19 (Impossibility of Plain Algebraic Signatures in Pairing Groups, with UF-CMA Security). *Let* SIG *be a plain algebraic signature scheme in pairing groups with parameters* $\lambda, k, \gamma \in \mathbb{N}$ *over a symmetric pairing group* $(\mathbb{G}, \mathbb{G}_T, e)$ *of (possibly) unknown order* N *and message space superpolynomial in* λ*. Then there exists a PPT adversary* $\mathcal{A}$ *with*

$$Adv_{\mathcal{A},\mathsf{SIG}}^{\mathsf{UF\text{-}CMA}}(\lambda) = 1$$

The adversary makes at most $Q_S = \gamma^2 \cdot |\tau_{\max}|$ *signature queries, where* $|\tau_{max}|$ *is the bitlength of the largest entry in the matrices* $A(hk, m)$ *for the messages that will be queried (we have* $|\tau_{\max}| = \mathrm{poly}(\gamma \log(N) + |m|)$*). In addition,* $|\mathcal{M}|$ *does not have to be superpolynomial, it suffices to require* $|\mathcal{M}| \geq Q_S + 1$*.*

Letting T_{linear} *denote an upper bound on the run-time of all invocations of the polynomial-time HNF algorithm and* $T_{\max}$ *be an upper-bound on the polynomial running time of functions* A*, our attack runs in* $O(Q_S \cdot (T_{\max} + T_{\text{linear}})) = O\Big(\gamma^2 \cdot |\tau_{\max}| \cdot (T_{\max} + T_{\text{linear}})\Big)$*.*

Proof Sketch. As in the proof of Theorem 14, the adversary iteratively obtains signatures for many messages $m_1 \ldots m_i$ and constructs an HNF matrix describing the column space generated by column vectors $A(hk, m_1)^\top \ldots A(hk, m_i)^\top$. Since $A(hk, m)$ is a row vector, the goal is to find a message m^* with $A(hk, m^*)^\top \in \mathsf{ColumnSpace}(A(hk, m_1)^\top | \ldots | A(hk, m_i)^\top)$ and to retrieve an integer vector $\boldsymbol{\alpha} \in \mathbb{Z}^i$, such that $A(hk, m^*) = \sum_{j=1}^i \alpha_j A(hk, m_i)$. The forgery signature $\mathbf{Y}^*$ is then computed as $\mathbf{Y}^* = \sum_{j=1}^i \alpha_j \mathbf{Y}_j$. Let us check that this indeed satisfies correctness:

$$e\Big(A(hk, m^*)^\top \cdot \mathbf{X}^{hk}, X\Big) = e\Big(\sum_{j=1}^i \alpha_j A(hk, m_i)^\top \cdot \mathbf{X}^{hk}, X\Big) =$$

$$\overset{\text{Correctness of } \mathbf{Y}_j}{=} \sum_{j=1}^i \alpha_j e(P, B(vk) \cdot \mathbf{Y}_j) = e(P, B(vk) \cdot \mathbf{Y}^*)$$

Arguing that the algorithm succeeds in forging after Q_S iterations is identical to the reasoning in Theorem 14.

BLS SIGNATURES. In the following, we will prove a result about the BLS signature scheme [8]. We will not formally define BLS signatures, since it will only be important which signatures are considered valid by BLS. In the BLS scheme, public keys are of the form $vk := X = x \cdot P \in \mathbb{G}$ for a group $\mathbb{G}$ of order p generated by P, and uniformly random $x \in \mathbb{Z}_p$. We also assume a hash function H, whose parameters may be added to vk if the function is parameterized. Valid signatures for a message m are of the form

$$\sigma = x \cdot \mathsf{H}(m).$$

BLS signatures consist of only one group element Y, and verification is performed by a pairing operation:

$$e(\mathsf{H}(m), X) = e(P, Y).$$

Boneh, Lynn, and Shacham [8] prove that if the used hash function H is modeled as a random oracle, then their scheme is $\mathsf{UF\text{-}CMA}$ secure under the computational Diffie-Hellman assumption in $\mathbb{G}$. In contrast, we prove that if H is algebraic (in the sense of Definition 17), then the scheme is insecure:

Theorem 20. *When implemented with an algebraic hash function* H*, the BLS scheme described above is* $\mathsf{UF\text{-}q\text{-}RMA}$*-insecure for a polynomial* $q = q(\gamma)$ *in the number of public group elements of* H.

To show Theorem 20, observe that BLS (when implemented with an algebraic hash function), is a plain signature in the sense of Definition 18. Hence, Theorem 20 follows from Theorem 19. Furthermore, if the order p of the used group G is prime, then tracing the steps of our attack actually shows that $q(\gamma) \leq \gamma + 1$.

Remark 21 (Waters Signatures). Note that Waters signatures [35] make use of programmable hash functions and symmetric pairing groups, and are known to be secure in the standard model. The attack in Theorem 19 does not extend to Waters signatures, because their verification equation pairs $\mathsf{H}(m)$ with parts of the signature, which is not allowed in Definition 18.

Remark 22 (Further generalization of plain and simplified algebraic signatures). We could also consider a definition of signature that captures the verification equations in both Definition 8 and Definition 18. By adjusting our attacks to concatenate the $A(hk, m)$ vectors from Definition 18 to the $\mathbf{c}(i)$ vectors of the attack in Theorem 14, one can obtain an attack against this slightly generalized signature class.

References

1. Masayuki Abe, Miguel Ambrona, Miyako Ohkubo, and Mehdi Tibouchi. Lower bounds on structure-preserving signatures for bilateral messages. In *Proc. SCN 2018*, volume 11035 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2018.

2. Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 649–666, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.
3. W.A. Adkins, S.H. Weintraub, J.H. Ewing, F.W. Gehring, and P.R. Halmos. *Algebra: An Approach Via Module Theory*. Graduate texts in mathematics. Springer-Verlag, 1992.
4. Boaz Barak and Mohammad Mahmoody-Ghidary. Lower bounds on signatures from symmetric primitives. In *48th Annual Symposium on Foundations of Computer Science*, pages 680–688, Providence, RI, USA, October 20–23, 2007. IEEE Computer Society Press.
5. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.
6. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.
7. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.
8. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.
9. Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Heidelberg, Germany.
10. Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. In Juzar Motiwalla and Gene Tsudik, editors, *ACM CCS 99: 6th Conference on Computer and Communications Security*, pages 46–51, Singapore, November 1–4, 1999. ACM Press.
11. Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 45–64, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.
12. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.
13. Michael A Frumkin. Polynomial time algorithms in the theory of linear diophantine equations. In *International Conference on Fundamentals of Computation Theory*, pages 386–392. Springer, 1977.

14. Rosario Gennaro, Yael Gertner, Jonathan Katz, and Luca Trevisan. Bounds on the efficiency of generic cryptographic constructions. *SIAM J. Comput.*, 35(1):217–246, 2005.
15. Michael Gerbush, Allison B. Lewko, Adam O'Neill, and Brent Waters. Dual form signatures: An approach for proving security from static assumptions. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 25–42, Beijing, China, December 2–6, 2012. Springer, Heidelberg, Germany.
16. Essam Ghadafi. Further lower bounds for structure-preserving signatures in asymmetric bilinear groups. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19: 11th International Conference on Cryptology in Africa*, volume 11627 of *Lecture Notes in Computer Science*, pages 409–428, Rabat, Morocco, July 9–11, 2019. Springer, Heidelberg, Germany.
17. Essam Ghadafi. Partially structure-preserving signatures: Lower bounds, constructions and more. IACR ePrint Archive, report 2020/477, 2020. `http://eprint.iacr.org/2020/477`.
18. Dennis Hofheinz, Tibor Jager, and Eike Kiltz. Short signatures from weaker assumptions. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 647–666, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.
19. Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 21–38, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.
20. Dennis Hofheinz, Eike Kiltz, and Victor Shoup. Practical chosen ciphertext secure encryption from factoring. *Journal of Cryptology*, 26(1):102–118, January 2013.
21. Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 654–670, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.
22. Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, WA, USA, May 15–17, 1989. ACM Press.
23. Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 2003: 10th Conference on Computer and Communications Security*, pages 155–164, Washington, DC, USA, October 27–30, 2003. ACM Press.
24. Kaoru Kurosawa and Yvo Desmedt. A new paradigm of hybrid encryption scheme. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 426–442, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.
25. Leslie Lamport. Constructing digital signatures from a one way function. Technical report, October 1979.
26. Ueli M. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete algorithms. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 271–281, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Heidelberg, Germany.

27. Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12, Cirencester, UK, December 19–21, 2005. Springer, Heidelberg, Germany.
28. Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany.
29. Daniele Micciancio and Bogdan Warinschi. A linear space algorithm for computing the hermite normal form. In *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation*, ISSAC '01, page 231–236, New York, NY, USA, 2001. Association for Computing Machinery.
30. Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *21st Annual ACM Symposium on Theory of Computing*, pages 33–43, Seattle, WA, USA, May 15–17, 1989. ACM Press.
31. John Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, MD, USA, May 14–16, 1990. ACM Press.
32. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.
33. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley Series in Discrete Mathematics & Optimization. Wiley, 1998.
34. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.
35. Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.
36. Mark Zhandry and Cong Zhang. The relationship between idealized models under computationally bounded adversaries. Cryptology ePrint Archive, Report 2021/240, 2021. https://eprint.iacr.org/2021/240.