UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Adapting Lattice-based Attacks to break Diffie-Hellman

Erweiterung von Lattice-Angriffen auf Diffie-Hellman

Masterarbeit

im Rahmen des Studiengangs
IT-Sicherheit
der Universität zu Lübeck

vorgelegt von
Jonah Heller

ausgegeben und betreut von
Dr. Sebastian Berndt

mit Unterstützung von
Thore Tiemann

Lübeck, den 9. Dezember 2022

Abstract

Lattices offer an extensive field of applications. One of which is developing new cryptographic procedures that are post-quantum secure or to prove security properties. Another application is to attack commonly used cryptographic protocols such as the Elliptic Curve Digital Signature Algorithm (ECDSA). Lattice-based attacks often exploit side-channel information to obtain partial knowledge of secret values. With the recent development of these attacks, the focus often lies on attacking ECDSA, while other protocols vulnerable are not considered. Therefore, we focused on adapting recent advancements to attack the Diffie-Hellman Key Exchange (DHKE).

Albrecht and Heninger showed that introducing non-linear knowledge, by including a predicate, can solve instances believed to be unattainable. While this approach increased the overall performance of the lattice-based attack on ECDSA, it does not apply to the default attacker model of DHKE. Therefore, we adapted the model to a more realistic scenario which enables to use of a partial-known-plaintext-attack to serve as a predicate suitable for the advanced lattice attack. As DHKE deviates from ECDSA in several aspects, e.g., the needed number of samples, we investigated and showed that the benefit of the ECDSA attack also applies to the DHKE attack. We also inspected the influence of the predicate and the different bit-sizes of DHKE on the timing behavior. Finally, we discussed possible mitigation and further advancements on the presented attack.

Zusammenfassung

Gitterstrukturen bieten eine Vielzahl von Anwendungsmöglichkeiten. Zum einen kann diese Struktur genutzt werden um neue kryptographische Verfahren, die sicher gegen Quantencomputer sind, oder um Sicherheitseigenschaften zu beweisen. Ein weiteres Anwendungsfeld von Gittern ist das Angreifen von kryptogaphischen Protokollen wie den Elliptic Curve Digital Signature Algorithm (ECDSA). Gitterbasierte Angriffe nutzen dafür häufig Information aus Seitenkanälen, um Teilinformationen über Werte, die einem Angreifer unbekannt sind, zu erhalten. Neuste Entwicklungen in diesem Feld konzentrieren sich hauptsächlich auf ECDSA, statt auch andere anfällige Systemen anzugreifen. Daher adaptieren wir einige dieser Verfahren, um auch Diffie-Hellman Key Exchange (DHKE) anzugreifen.
Albrecht und Heninger zeigten, dass das Einbinden von nicht linearem Wissen, mittels eines Prädikates, Instanzen lösen konnte, die bis dahin als nicht lösbar galten. Während dieser Ansatz die Performance von dem Angriff auf ECDSA erhöhte, ist das Prädikat nicht direkt auf DHKE anwendbar. Deshalb erweitern wir das Standard-Angreifer-Modell, sodass wir einen partial known plaintext Angriff nutzen, um das Prädikat zu erhalten. Da sich DHKE und ECDSA zum Beispiel in der Dimension unterscheiden, haben wir diesen und weitere Unterschiede untersucht und konnten zeigen, dass sich ähnliche Leistungssteigerungen für DHKE im Vergleich zu ECDSA beobachten lassen. Außerdem haben wir den Einfluss des Prädikats und die verschiedenen Bit-Größen von DHKE auf ihr Zeitverhalten untersucht. Abschließend diskutieren wir mögliche Gegenmaßnahmen sowie zusätzliche Erweiterungen für den betrachteten Angriff.

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Lübeck, 9. Dezember 2022

Acknowledgements

I would like to thank Sebastian and Thore for mentoring me during writing this thesis and answering my questions. Thanks also to Anja and Daniel for helping me to elevate this thesis into a readable state. Further thanks to my family and friends for their continuous support. Finally, I want to thank my janitor, who gave his best to fix all the pipe ruptures in my apartment as fast as possible.

Contents

1 Introduction

Lattices are mathematical structures which serve as foundation for cryptographic schemes secure against quantum computers. They can also be used as a basic element to attack certain cryptographic schemes. When utilized to attack schemes it is combined with secret information, which in most cases is supplied by leakage obtained using side-channel attacks. The general concept for either building and breaking cryptographic system with lattices are their hard problems. While such hard problems serve as security barriers for newly developed post-quantum secure systems, they can be exploited to leverage commonly used systems. One post-quantum secure scheme that originates from this concept is NTRU [HPS98], which was further developed and committed as a NIST post-quantum cryptographic system candidate.

Applying lattices to attack systems has gained more attention over the last decade. But recent advancements on lattice-based attacks mainly focused on the Elliptic Curve Digital Signature Algorithm (ECDSA), even though schemes such as the Diffie-Hellman Key Exchange (DHKE) and RSA are also threatened by similar attacks. Therefore, this thesis is focused on elaborating on how certain schemes such as ECDSA can be attacked using lattices and which extensions are available. We give a step-by-step construction for the basic lattice-attack on ECDSA and adapt it to the lattice attack on DHKE. As recent enhancements are focused on attacking ECDSA, we also study the applicability on DHKE.

The DHKE is widely used in communication to exchange a secret between two or more parties. It has found application in a variety of protocols, which leads to many possibilities for leakages through flaws in implementations. A use-case of DHKE is for TLS communication, where the shared secret is used for symmetric encrypted communication.

When comparing lattice attacks on ECDSA to DHKE one will find that lattice attacks are more viable on ECDSA. Nevertheless, as DHKE is widely used, it must be considered vulnerable by lattice attack adaptations as well. Therefore, this thesis applies recent adaptations to the DHKE scheme to show that these adaptations can be applied on different schemes as well.

1.1 Related Work

The main adaptation used for this work is based on "On Bounded Distance Decoding with Predicate: Breaking the 'Lattice Barrier' for the Hidden Number Problem" [AH21], which

introduced a new technique to solve even more difficult instances by adding a predicate to the attack. With the introduction of this predicate, the attack can find correct solutions in cases where previous methods could not have been applied. The predicate can include even non-linear relations of parameters, as it only needs to be verified.
As [AH21] points out, there is also another approach to refine the lattice attack, by including hints [DDGR20]. These hints are linear knowledge, which are embedded into the lattice basis to increase the performance of the lattice attack. There are several key differences between hints and predicates: hints can be used to embed more linear knowledge into the lattice to enhance its results, while the predicate can add even non-linear knowledge to the basic problem of lattice reduction. Another difference is the impact on the attack results, as the predicate adaptation can solve even more difficult instances than the hint adaptation. While the hints are embedded during construction of the lattice, the predicate is only applied on solving the core problem of lattice reduction. The approach of [AH21] cannot be used directly with simple lattice reduction algorithms, as the predicate needs to be implemented in the solving process of the reduction.

1.2 Motivation

Adapting lattice-based attacks with the approach of [AH21] showed that even barriers which were widely believed to be out of scope could be broken. As this adaptation was only applied to ECDSA, where the introduction of a strong predicate is trivial, it raises interest in other techniques resembling DHKE. Combining DHKE with a predicate in its typical attacker model is not possible, as no additional knowledge is given. In a specific scenario, a predicate for DHKE can be introduced, and therefore the lattice attack adaptations mentioned may be applied. Unlike ECDSA, DHKE builds on a completely different context and might include different computational overhead regarding the predicate. Therefore, this thesis focuses on testing this adaption to DHKE by evaluating the core techniques of the implementation of [AH21]. Furthermore, we evaluate the impact of the number of samples and the predicate on the time behavior of the solving algorithms.

2 Preliminaries

In this chapter, basic concepts are described, including lattices, their applications as well as Elliptic Curve Digital Signature Algorithm (ECDSA) and Diffie-Hellman Key Exchange (DHKE) and their lattice-based attacks. ECDSA and DHKE are the main cryptographic schemes described and used in this thesis. Both schemes require multiple symbols for their description and the construction of an attack. Therefore, the schemes may share similar notation, but the symbols are always to be seen in the context of the considered scheme.

There are various lattice-based attacks on systems like RSA [MH20], ECDSA [AFG$^+$14, MSEH20, AH21] and DHKE [BV96, MBA$^+$21] focusing on recovering private values or keys using linear combinations of the lattice basis. The core principle and its adaptations are described further in this chapter. Lattice attacks often use knowledge about some hidden or private values, which might be presented through leakage revealed by side-channel attacks. This leakage can be as trivial as short timing differences in computation, which might leak a few leading bits of a hidden value in computation. Several leaked bits are often sufficient to recover the complete value or even other private values, as recent side-channel attacks show [MSEH20, AFG$^+$14].

We first explain the schemes ECDSA and DHKE as well as the mathematical lattice structure. The research on lattices introduces new hard problems, that are vulnerable to lattice attacks themselves. Then we explain the construction of a basic lattice attack on ECDSA by reducing different lattice problems to each other. Finally, we discuss advanced lattice reduction techniques such as sieving and enumeration as well as advanced lattice-based attacks.

2.1 Elliptic Curve Digital Signature Algorithm (ECDSA)

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a signature algorithm used to sign data and verify signatures. The ECDSA parameters are standardized by NIST [oST13]. It uses an elliptic curve $\mathcal{EC}(\mathbb{F}_q)$ over a finite field $\mathbb{F}_q$ and a generator point G of order n. To compute a signature for message M, the private signing key d, the hash digest $h = \mathrm{hash}(M)$ and a random nonce k are used to compute the pair (r, s) where r is the x-coordinate of $R = (k \cdot G)$ and

$$s \;=\; k^{-1} \cdot (h + d \cdot r) \bmod n. \tag{2.1}$$

By using the public verification key $Q = d \cdot G$, the message M and the tuple (r, s), anyone can verify the signature. For verification one needs the hash digest h, generator G, signature (r, s) and the public verification key Q to verify $r = r'$ where r' is the x-coordinate of $R' = (h \cdot s^{-1}) \cdot G + (r \cdot s^{-1}) \cdot Q$.

2.1.1 Leakage

Scalar multiplication with elliptic curve points is successive point additions over the elliptic curve. These successive additions are computationally expensive compared to simple multiplications in multiplicative groups. Therefore, the calculation of R is substantially more expensive than calculating s. This significant difference makes measuring the timing behavior of the computation R feasible which results in leakage of the most significant zero bits of k. Timing behavior is one key vulnerability for leakage through timing attacks [MSEH20]. Another problem with extensive computation lies in power requirement resulting in another possible side-channel [AFG+14] i.e. a power consumption side-channel. In the specific case of signing using ECDSA, side-channel attacks can leak knowledge of k through the computation of R. Timing leakage, for example, might reveal some of the most significant bits of k, as the preferred non-constant time algorithms used to compute R might terminate earlier if k is small. By recovering the nonce value k, the private signing key can be recovered by rearranging Equation 2.1 for calculating s to d:

$$d = (s \cdot k - h) \cdot r^{-1} \bmod n$$

2.2 Diffie-Hellman Key Exchange (DHKE)

The Diffie-Hellman Key Exchange's (DHKE) purpose is to securely share a secret between two parties, here Alice and Bob. Each party chooses a secret value, which is used to compute a public value shared with the other party. To compute the shared secret, each party combines the public value of the respective other party with its private value. The shared secret is often used to derive a symmetric encryption key for further secure communication between the two parties.
In our case, DHKE uses a prime number p and generator g for the multiplicative group $\mathbb{Z}_p^*$, that are publicly known. As Figure 2.1 shows, each party will choose a random private value a, b respectively. These values are used to compute the public values A and B which

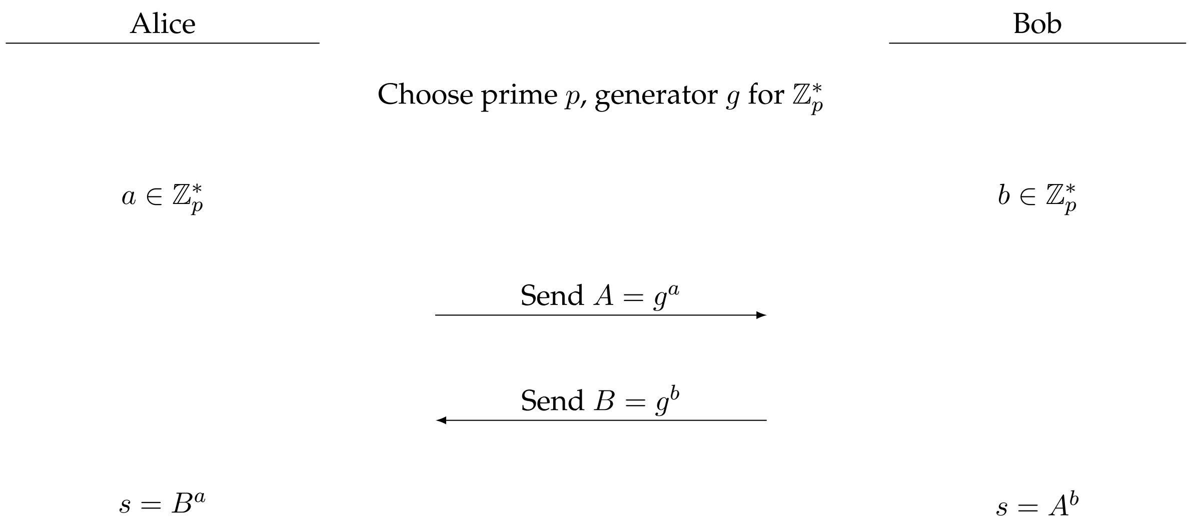

Figure 2.1: Diffie-Hellman Key Exchange (DHKE) between two parties referred to as Alice and Bob. Both parties will randomly sample a secret value. Dependent on the use-case one party might want to reuse its private value multiple times.

are exchanged. Thus, the first party (Alice) can obtain the shared secret $s = B^a = (g^b)^a = g^{(a \cdot b)}$, while the second party (Bob) calculates $s = A^b$. As both calculations result in the same value, the key exchange is complete.
In general, DHKE can be defined for any finite cyclic group, following the general steps similar to the multiplicative group $\mathbb{Z}_p^*$. Elliptic Curve Diffie-Hellman (ECDH), for example, uses an elliptic curve with a generator point and is also commonly used in many security protocols such as TLS. This work focuses on attacking DHKE over the multiplicative group $\mathbb{Z}_p^*$ using lattice attacks. As of writing the applicability of lattice attacks on ECDH is unknown.

2.2.1 Security Considerations

For any attacker, the only known values are p, g, A and B. One way to recover the shared secret is to acquire one of the private values. If an attacker only has the value A respectively to recover a, the discrete logarithm of A to g, i.e. $\mathrm{dislog}_g(A)$ needs to be calculated. To the best of our knowledge, it is unknown whether the discrete logarithm is hard or efficiently computable disregarding the use of quantum computers [Sho94] and if it is the only possible strategy to compute the private value a for a given public value A. To describe security for Diffie-Hellman on general cyclic groups, the Decisional Diffie-Hellman (DDH) and Computational Diffie-Hellman (CDH) assumptions are defined as follows:

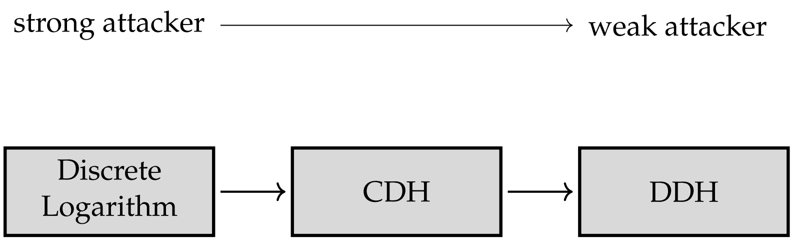

Figure 2.2: Security for DHKE for general cyclic groups.

Definition 2.1 (Decisional Diffie-Hellman (DDH)). The DDH assumption asserts that the probability distribution for the two tuples $(g^a, g^b, g^{a \cdot b})$ and (g^a, g^b, g^c), with a, b, c chosen randomly and independently from $\mathbb{Z}_q$, are computationally indistinguishable. If an attack cannot distinguish the tuples in polynomial time, the assumptions hold.

Definition 2.2 (Computational Diffie-Hellman (CDH)). The CDH assumption expresses that computing $g^{a \cdot b}$, given g, g^a and g^b with randomly chosen $a, b \in \mathbb{Z}_q$ and randomly chosen generator g, is computationally intractable. If an attack cannot compute $g^{a \cdot b}$ in polynomial time the assumption holds.

As Figure 2.2 displays, the security for DHKE over multiplicative groups $\mathbb{Z}_p^*$ is implied by the discrete logarithm as the basic building block. If an attacker can solve the discrete logarithm problem, the CDH can easily be calculated. The CDH assumption is stronger than the DDH assumption, as any attacker solving the CDH can solve the DDH as well. By solving only the DDH, it is not known whether an attacker can implicitly solve the CDH.
There are algorithms, such as the Pohlig-Hellman algorithm [PH78], which can be used to compute the discrete logarithm for relatively large prime numbers. To defend systems against such algorithms, one must choose a safe and large prime number. A prime number p is considered safe, if it fulfills $p = 2 \cdot p' + 1$, with p' being prime as well. In current standards for TLS, the DHKE is used with 2048-bit and 3072-bit large prime numbers p.

2.3 Lattices and Applications

Lattices are applicable in cryptography for either recovering secret values and breaking cryptographic systems or for building cryptographic systems by exploiting properties from hard problems in lattices.

Definition 2.3 (Lattice). The lattice $\Lambda = \left\{ \sum_{i=1}^{d} a_i v_i \middle| \, a_i \in \mathbb{Z} \right\}$ is a subgroup of $\mathbb{R}^d$ and is defined as a set of linear independent vectors $v_1, \ldots, v_d$ with $v_i \in \mathbb{R}^{d'}$ where $d' \geq d$.

As related works using lattices often utilize the row representation for the lattice basis, all following basis matrices will be given in row representation as well. Furthermore, norms regarding lattices are denoted by $|| \cdot ||$ and should be interpreted always as euclidean. The smallest ball centered at the origin of the lattice containing i linear independent vectors has the radius of $\lambda_i(\Lambda)$. Hence $\lambda_1(\Lambda)$ contains the shortest non-zero vector in Λ.
In lattice structures, solving certain problems is much more difficult than in $\mathbb{R}^d$. Furthermore, the hardness of these problems is directly dependent on their dimension. If the dimension is in a certain computational size, even hard lattice problems can be solved, which brings the opportunity to attack certain cryptographic systems. As mentioned in Section 2.1, there exist multiple attacks on ECDSA using lattice-based attacks to recover the private signing key. There are also lattice-based attacks on DHKE [MBA+21] and RSA [May03].
In the following, we provide the general problem definitions, which are further used for the lattice construction of the attacks.

2.4 Lattice Problems

As lattices are only a subgroup of $\mathbb{R}^d$, it introduces hard problems like the Shortest Vector Problem (SVP) or Closest Vector Problem (CVP). Both are used to either conceive a problem, which is computationally infeasible to solve, or that can be exploited by solving a feasible problem instance to recover secret values. In Section 2.5, we will discuss an algorithm (Algorithm 1) used to solve the SVP problem. The runtime complexity of this algorithm scales by the dimension of the lattice which directly relates to the feasibility of problem instances.
The following problem definitions are based on [AH21] because our implementation is based on their prior work. Therefore, definitions might vary from other literature.

Definition 2.4 (Closest Vector Problem (CVP)). Given a lattice Λ and a target vector $v_{\text{target}} \in \mathbb{R}^d$, find the closest lattice point, e.g. an integer combination of lattice vectors $v_1, \dots, v_d$ that is the closest to v_{target}.

Definition 2.5 (Shortest Vector Problem (SVP)). Given a lattice Λ find the shortest non-zero vector $v \in \Lambda$.

As the SVP means to search for the shortest vector $v \in \Lambda$, it can be solved by orthogonalizing the basis B of Λ to B'. The orthogonal basis B' then consists of the smallest possible vectors, which are relatively orthogonal. Thus, B' will contain a vector v with a norm of $||v|| = \lambda_1(\Lambda)$. Because the norm of two distinct vectors of the lattice can be equal, and $\lambda_1 \leq \lambda_i$ for $i \in 2, \dots, d$, the SVP can have more than one solution.

Definition 2.6 (α-Bounded Distance Decoding (BDD$_\alpha$)). The distance $\mathrm{dist}(v, \Lambda)$ describes the difference between a vector v and the closest lattice point of the lattice Λ. Given a lattice Λ, the target vector t and some parameter $\alpha > 0$ where $\mathrm{dist}(t, \Lambda) < \alpha \cdot \lambda_1(\Lambda)$, find the vector $v \in \Lambda$ with distance to t is shortest of all $v' \in \Lambda$. Thus the distance of t to Λ is bounded by α and the radius of the smallest ball of Λ.

The definition of BDD$_\alpha$ leads to a bound of α which determines that t must be close to one point of Λ. As the SVP lattice problem described above does not necessarily have a unique solution, the following definition provides a problem formulation with a unique solution, which is essential to set up lattice-based attacks.

Definition 2.7 (γ-unique Shortest Vector Problem (uSVP$_\gamma$)). Find the shortest vector $v \in \Lambda$ (c.f. Definition 2.5), where $\lambda_2(\Lambda) > \gamma \cdot \lambda_1(\Lambda)$.

Definition 2.7 states, that λ_2 needs to be greater than Λ_1 by the factor of γ and by $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_d$ there can be only one vector with radius λ_1. Hence, one can find a "unique" solution for uSVP$_\gamma$. This offers the possibility to construct an instance for an attack, where the unique solution reveals the recovered secret value.

2.5 Lattice Attacks

Lattice-based attacks mostly follow the principle of constructing a problem instance, which is further reduced to a CVP instance, to then be solved using lattice reduction algorithms. More precisely, one collects data and forms a Hidden Number Problem (HNP, c.f. Definition 2.8) instance, which will be solved to find a private value or secret key. The collected data will most likely consist of publicly known values as well as some leaked partial knowledge of secret values. To solve the HNP, a reduction to a lattice-based problem such as CVP or SVP is used. As the problem instance will most likely fulfill a boundary for the target vector, the BDD$_\alpha$ problem is used as a more specific version of the CVP. By solving a BDD$_\alpha$ instance, one will not receive the solution directly, as the solution is presented as the difference between the target vector and the solution vector. Therefore, one reduction step to the SVP or uSVP$_\gamma$ is appended to the procedure. Reducing onto the SVP includes the solution vector in the lattice and the solution to the initial problem can thus be found directly.
The reduction chain from Figure 2.3 may be seen as the standard procedure for lattice-based attacks. The following sections will review all construction and reduction steps in more detail using a lattice-based attack on ECDSA. Constructing the lattice for the lattice-based attack on DHKE shares all those steps and optimizations as well.

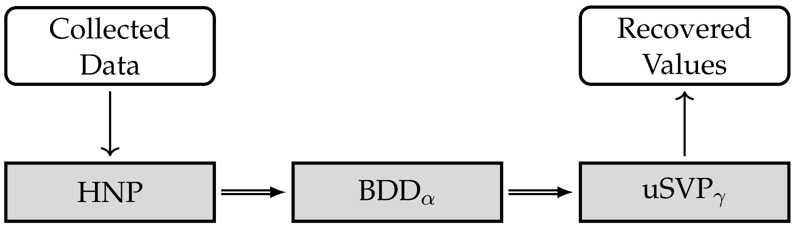

Figure 2.3: Reduction steps from initially collected data up to the resulting recovered values.

Algorithm 1: LLL-Reduction

```
   Input: Basis $B$ for lattice $\Lambda$, where $B = \mathrm{span}(v_0, \dots, v_n)$
   Output: Reduced basis $B'$ for lattice $\Lambda'$
 1 $B^* \leftarrow \mathrm{GramSchmidt}(\{v_0, \dots, v_n\}) = \{v_0^*, \dots, v_n^*\}$
 2 $\mu_{i,j} \leftarrow \frac{v_i \cdot v_j^*}{v_j^* \cdot v_j^*}$
 3 $k \leftarrow 1$
 4 while $k \leq n$ do
 5     for $j \in \{k-1, \dots, 0\}$ do
 6         if $|\mu_{k,j}| > \frac{1}{2}$ then                          // Size reduction
 7             $v_k \leftarrow v_k - \lfloor \mu_{k,j} \rceil \cdot v_j$
 8             $B^* \leftarrow \mathrm{GramSchmidt}(\{v_0, \dots, v_n\}) = \{v_0^*, \dots, v_n^*\}$
 9     if $v_k^* \cdot v_k^* > (\delta - \mu_{k,k-1}^2) \cdot v_{k-1}^* \cdot v_{k-1}^*$ then     // Lovász condition
10         $k \leftarrow k + 1$
11     else
12         $v_k \leftarrow v_{k-1}$
13         $k \leftarrow \max(k-1, 1)$
14 return $B$
```

Solving SVP

To find the solution to the SVP, the lattice basis Λ has to be orthogonalized as commonly done via the Gram–Schmidt process for Euclidean spaces ($\mathbb{R}^n$) with scalar values in $\mathbb{R}$. As the Gram–Schmidt process is using a projection for scalar multiplication of vectors defined on $\mathbb{R}$, the resulting scalar values of the process can simply be rounded regarding $\mathbb{Z}$. This process has polynomial time complexity for dimension $n = 2$, but for dimensions greater than $n = 2$, the process is believed to be hard. Reducing lattice basis matrices with a large dimension for an optimal solution leads to exponential computation time. The adaptation of the Gram–Schmidt process is called the Lenstra–Lenstra–Lovász (LLL) [AKLL82] algorithm.

Algorithm 1 shows the general structure of the LLL algorithm, which includes two important parts: Size reduction and the Lovász condition. The size reduction can be seen in

lines 5 and 6, which reduces the vector v_k by each v_j. The vector v_k can only be reduced by v_j if $|\mu_{k,j}| > \frac{1}{2}$ holds. Furthermore, the Lovász condition in line 9 affects the reduction of the vector v_k. Hence, the vector v_k is swapped to be reduced further if the reduction was not sufficient. As one can see, the algorithm terminates if each vector fulfills the Lovász condition. Therefore, the vectors are arranged according to their size. The parameter δ defies the resulting precision of the algorithm and has to be within the interval of $\frac{1}{4}$ to 1. If δ is 1, the algorithm will find the optimal reduced lattice basis, but the algorithm might not terminate in polynomial time. Any value for δ below 1 will result in polynomial execution time. Commonly δ is set to $\frac{3}{4}$, so that the algorithm computes a solution in polynomial time and the shortest vector b_1 obtained by the algorithm fulfills $|b_1| \leq 2^{\frac{(n-1)}{2}} \cdot \lambda_1(\Lambda)$. If the basis to be reduced is well constructed, one can reliably find the shortest vector.
Further techniques such as sieving and enumeration used to solve the SVP are described in Section 2.6.

2.5.1 Hidden Number Problem (HNP)

The Hidden Number Problem (HNP) introduced by [BV96], is a suitable framework to construct a lattice-based attack. In [BV96] the goal was to formulate a problem instance created for specific information to the DHKE. As the HNP is the basic construction for lattice-based attacks, various derivatives suit the specific problem set-up.

Definition 2.8 (Hidden Number Problem). Given prime p, length $L \in \mathbb{N}$ and n tuples (t_i, a_i), find the secret $\alpha \in \mathbb{F}_p$, where for every tuple there exists a $k_i \in \{1, \ldots, L-1\}$ such that $a_i + k_i = (t_i \cdot \alpha) \mod p$.

As Definition 2.8 shows, the HNP consists of n values k_i, which are relatively small compared to L. The property of small values of k_i will later be used to find the solution by solving its SVP instance. One can construct a lattice so that there exists a vector containing all values for k_i in the lattice. Because the values for k_i are small, the norm of their vector is small as well. With this property, the short vector can be found by solving the SVP.

ECDSA as HNP

For lattice attacks, small values are needed because we construct an HNP instance exploiting these small values. As discussed in Section 2.1 the nonce parameter k of ECDSA is vulnerable to side-channel attacks, which might lead to leakage of most significant bits. Most side-channel attacks leak data by measuring computation time differences, which results in leaked 0 bits as the most significant bits. Thus, the unknown bits of k result in a smaller value, making lattice attacks feasible. To build the lattice-based attack, signatures

originating from short nonces need to be collected. The signatures then are used to build equations to construct an HNP instance.
With known signatures, the attacker has the tuple (r, s), the message m and its hash h. To build the HNP instance one needs to construct equations as described in Definition 2.8:

$$\begin{aligned} s &\equiv k^{-1} \cdot (h + d \cdot r) \mod n & (2.2)\\ k &\equiv s^{-1} \cdot h + s^{-1} \cdot d \cdot r \mod n & \\ -s^{-1} \cdot h + k &\equiv s^{-1} \cdot r \cdot d \mod n. & (2.3)\end{aligned}$$

The color code used in the equations above highlight known values (blue) and unknown values (red).
Given the description for ECDSA in Equation 2.2, one can rearrange it to Equation 2.3. If we label Equation 2.3 with the Labels 2.4, we will receive the equation form of the basic HNP (see Definition 2.8):

$$\begin{aligned} a_i + k &\equiv t_i \cdot \alpha \mod n \\ a_i = -s_i^{-1} \cdot h_i,\ \ t_i &= s_i^{-1} \cdot r_i,\ \ \alpha = d. \qquad (2.4)\end{aligned}$$

Note that one can use multiple equations as Definition 2.8 states. As multiple equations are needed to solve instances with fewer known bits, we further define one equation with its related knowledge as samples. Considering ECDSA, one sample consists of the hash value h_i for the data to sign as well as the related signature tuple (r_i, s_i).

2.5.2 HNP to CVP

In this section, the general idea of the transformation from an HNP instance to a CVP instance is described.
As the vector $v \in \Lambda$ is closest to the target vector v_{target}, the basis of the lattice can be constructed, to exploit the fact that the difference between v and v_{target} is small. Therefore, the small values k_i from the HNP instance should be the difference between both vectors. The value of α is not known, so the values t_i should be included in the lattice basis. Thus, the values for a_i are used as the target vector. By solving the CVP instance the vector $v' = (t_1, \dots, t_m, \frac{1}{n}) \cdot \alpha$ may be computed and as the target vector is $v_{\text{target}} = (a_1, \dots, a_m, x)$, the resulting difference is $v_{\text{diff}} = (k_1, \dots, k_m, \frac{\alpha}{n} - x)$, with $x \in \mathbb{R}$. Since the values k is relatively small, the probability that v' is selected as the solution is high.

As Boneh and Venkatesan described in [BV96], a hidden number instance can be embedded into the lattice

$$\Lambda = \begin{pmatrix} n & 0 & 0 & \dots & 0 & 0 \\ 0 & n & 0 & \dots & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & 0 & \dots & n & 0 \\ t_1 & t_2 & t_3 & \dots & t_m & \frac{1}{n} \end{pmatrix}$$

which conveniently contains the vector

$$v' = \left(t_1 \cdot \alpha \bmod n, \dots, t_m \cdot \alpha \bmod n, \frac{\alpha}{n}\right),$$

given by the linear combination:

$$v_{lc} = (x_1, \dots, x_n, \alpha),$$

with $x_1, \dots, x_n \in \mathbb{Z}$ as $x_i \cdot n \equiv 0 \bmod n$. With high probability, v' is closest to

$$v_{\text{target}} = (a_1, \dots, a_m, 0),$$

and their difference $v_{\text{diff}} = v' - v_{\text{target}} = (k_1, \dots, k_m, \frac{\alpha}{n})$ reveals all unknown values for k. Thus, this embedding solves the HNP instance, as any value k_i can be used to compute

$$\alpha = (a_i + k_i) \cdot t_i^{-1} \mod p.$$

2.5.3 CVP to SVP

Because solving the SVP outputs the shortest vector of the lattice basis Λ, it is our goal to include the vector $v_k = (k_1, \dots, k_m, \frac{2^L \cdot \alpha}{n}, 2^L)$. We can construct a basis, with $v_k \in \Lambda$, by embedding the target vector in the lattice to ensure that there exists a linear combination resulting in v_k. The values for k_i are small by definition. Therefore, the vector v_k should also be relatively small and is a good candidate as an element of the orthogonalized basis.

Using Kannan's embedding [Kan87], the CVP instance can be converted to an SVP instance. We simply add the target vector of the CVP instance into Λ:

$$\Lambda' = \begin{pmatrix} n & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & n & 0 & \dots & 0 & 0 & 0 \\ & \vdots & & & \vdots & & \\ 0 & 0 & 0 & \dots & n & 0 & 0 \\ t_1 & t_2 & t_3 & \dots & t_m & \frac{2^L}{n} & 0 \\ a_1 & a_2 & a_3 & \dots & a_m & 0 & 2^L \end{pmatrix}$$

This lattice contains the vector

$$v_{\text{sol}} = \left(k_1, \dots, k_m, 2^L \cdot \frac{\alpha}{n}, -2^L\right),$$

which is constructed by the linear combination:

$$v_{lc} = (x_1, \dots, x_n, \alpha, -1),$$

where $x_i \in \mathbb{Z}$. Hence, the vector v_{sol} will be short regarding its norm , as

$$||v_{\text{sol}}|| \leq \sqrt{m+2} \cdot 2^L.$$

With the linear combination and the lattice matrix the solution vector solving the HNP can be found, according to Equation 2.3. Therefore, we will simply rearrange Equation 2.3 with the Labeling 2.4 to Equation 2.5:

$$a_i + k_i - t_i \cdot \alpha \equiv 0 \bmod n. \tag{2.5}$$

The linear combination v_{lc} then results in v_{sol} because $x_i \cdot n \equiv 0 \bmod n$. For any k_i one can compute the rearranged Equation 2.3 with

$$\begin{aligned} n \cdot x_i + t_i \cdot \alpha - a_i \cdot 1 &\equiv t_i \cdot \alpha - a_i \bmod n \\ t_i \cdot \alpha - a_i &\equiv k_i \bmod n. \end{aligned}$$

The vector v_{sol} will then hold the unknown values for k, which can be used to recover d. As we use the CVP to solve this instance, we have to make sure that the lattice does not contain any shorter vectors other than v_{sol}.

In this construction as is, the vector v_{sol} will not be the shortest, as Λ' also contains the vector $v_{2^L} = (0, \ldots, 0, 2^L, 0)$. The vector v_{2^L} is in the lattice due to the linear combination $v_{lc'} = (-t_1, \cdots, -t_m, n, 0)$. Thus, the lattice needs to be adapted by rearranging Equation 2.3.

Construction without α

In previous constructions, the value α was applied to the lattice to recover the values for k_i. As all k_i are concerning the same α, one simply can eliminate this variable from the construction. If one has recovered k_i the value $d = \alpha$ also can be recovered. The following construction is analogous to [ACPS09].
By combining Equations 2.5, the value α can be simply eliminated. Therefore, one then selects one of the m distinct equations. The selected equation is labeled with an index of 0 and the rest is labeled with indices $i = 1, \ldots, m-1$:

$$\begin{aligned}
a_i + k_i &\equiv t_i \cdot \alpha \bmod n \\
(a_i + k_i) \cdot t_i^{-1} &\equiv \alpha \bmod n \\
(a_i + k_i) \cdot t_i^{-1} &\equiv (a_0 + k_0) \cdot t_0^{-1} \bmod n \\
a_i + k_i &\equiv t_0^{-1} \cdot t_i \cdot a_0 + k_0 \cdot t_0^{-1} \cdot t_i \bmod n \\
a_i - t_i \cdot t_0^{-1} \cdot a_0 + k_i &\equiv t_i \cdot t_0^{-1} \cdot k_0 \bmod n. \qquad (2.6)
\end{aligned}$$

One ends up with $m-1$ equations in the form of Equation 2.6. With the labeling of $a'_i = a_i - t_i \cdot t_0^{-1} \cdot a_0$ and $t'_i = t_i \cdot t_0^{-1}$ one again obtains the structure of the HNP with

$$a'_i + k_i \equiv t'_i \cdot k_0 \bmod n.$$

Therefore, the α value for the HNP instance is no longer d, but the nonce value k_0.
To recover all values for k_i we need to change the value of $\frac{2^L}{n}$ of Λ' to 1. With this change and the new values t'_i and a'_i, we construct the lattice

$$\Lambda'' = \begin{pmatrix}
n & 0 & 0 & \ldots & 0 & 0 & 0 \\
0 & n & 0 & \ldots & 0 & 0 & 0 \\
 & \vdots & & & \vdots & & \\
0 & 0 & 0 & \ldots & n & 0 & 0 \\
t'_1 & t'_2 & t'_3 & \ldots & t'_{m-1} & 1 & 0 \\
a'_1 & a'_2 & a'_3 & \ldots & a'_{m-1} & 0 & 2^L
\end{pmatrix}.$$

This lattice will contain the vector $v_{\text{sol}} = (k_1, \ldots, k_{m-1}, k_0, -2^L)$, which can be constructed

using the linear combination $v_{lc} = (x_1, \dots, x_{m-1}, k_0, -1)$.
By replacing $\frac{2^L}{n}$ with 1 one does not only filter the value of k_0, but also prevent Λ'' from containing any non-trivial vector of the form $(0, \dots, 0, *, 0)$. When choosing a linear combination similar to $v_{lc'}$ the result is the trivial vector $v_0 = (0, \dots, 0)$, which cannot be a basis vector, as it is not linear independent of any other vector of $\mathbb{R}^n$. Thus, the shortest vector in Λ'' is expected to be v_{sol}.

1-Bit Reduction

In [NS03], the authors present another improvement to the construction of the lattice. This improvement reduces the target values of the target vector by 1 bit, without needing any further information for the HNP instance.
Considering ECDSA, where the values of the target vector are given by the values of k_i with $k_i < 2^L$, one also knows that every $k_i \geq 0$. This simple fact gives the opportunity to shift the target vector by 2^{L-1}. Shifting the target vector by that amount will guarantee, that $k_i \in [-2^{L-1}, 2^{L-1}]$ instead of $k_i \in [0, 2^L]$. This reduces the norm of the target vector from $||v_{\text{target}}|| \leq \sqrt{m+2} \cdot 2^L$ to $||v_{\text{target},1-\text{bit}}|| \leq \sqrt{m+2} \cdot 2^{L-1}$. To adapt Λ'' with that improvement, only one value needs to be updated, which results in

$$\Lambda''_{1-\text{bit}} = \begin{pmatrix} n & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & n & 0 & \dots & 0 & 0 & 0 \\ & \vdots & & & \vdots & & \\ 0 & 0 & 0 & \dots & n & 0 & 0 \\ t'_1 & t'_2 & t'_3 & \dots & t'_{m-1} & 1 & 0 \\ a'_1 & a'_2 & a'_3 & \dots & a'_{m-1} & 0 & 2^{L-1} \end{pmatrix}.$$

Note that reducing the norm of the target vector results in a better performance as described in [NS03], but will also change the target vector slightly. The new resulting target vector

$$v_{\text{target},1-\text{bit}} = (k_1 - 2^{L-1}, \dots, k_{m-1} - 2^{L-1}, k_0 - 2^{L-1}, 2^{L-1})$$

is shifted by exactly 2^{L-1} and needs to be lifted again to obtain the real results for k_i.

Solving SVP

To solve the SVP instance, one can use the LLL algorithm, as it orthogonalizes the lattice basis. By definition, the orthogonalized basis will consist of short and nearly orthogonal vectors. As the vector v_k is short, the probability is high that the orthogonalized basis contains this vector.

More advanced strategies such as lattice enumeration and lattice sieving are discussed in the next Section 2.6.

2.6 Advanced Lattice Techniques

There are more advanced techniques to find a certain shortest vector in a lattice than only using the previously mentioned LLL algorithm. As LLL orthogonalizes the basis of the lattice, it will most certainly find the shortest vector. If the target vector is not the shortest vector and is only slightly longer than the shortest basis vector, more advanced techniques are needed. Another problem might be that the algorithm does not find a basis containing the shortest vector. In that case, a search algorithm is needed, which takes an orthogonalized basis and tries to find vectors of the lattice fulfilling the target.

In the following chapter, we will adapt DHKE to the strategies and implementation of [AH21]. The former work includes two main search algorithms, namely sieving and enumeration.

2.6.1 Sieving

Sieving as proposed in [AKS01] and its adaptations [MV10, BGJ15, Laa16, BDGL16, HK17], takes a set of lattice vectors $L \subset \Lambda$ and searches for a combination of those vectors to find a short vector. The general idea is to take exponentially many vectors of the lattice and combine k vectors by taking their sums or differences at a time to obtain a shorter vector. An algorithm taking k vectors at a time is called k-sieve. Figure 2.4 shows this procedure with n steps. As the figure describes, the sieve technique generates a set of vectors V_1 for a given lattice Λ. Then this set is used with a k-sieve to generate a new set of vectors V_2, which holds fewer vectors than V_1. Each new vector in V_2 is shorter than the vectors of V_1. This procedure is repeated n times to reduce the number of vectors, while also decreasing the lengths of the vectors. Thereby, after n steps, the resulting set of vectors is significantly smaller and all vectors of this set are substantially shorter than the vectors before. To make sure that each round reduces the set of vectors to shorter vectors, a radius R_i can be defined for round i to bind the vectors to be accepted, because each vector in the reduced set must fulfill the radius boundary for the specific round. Note that this procedure does not find the shortest vector but a set of arguably short vectors within a radius of R_n. This is to find a solution that is known to be in radius R but is not the shortest vector of the lattice.

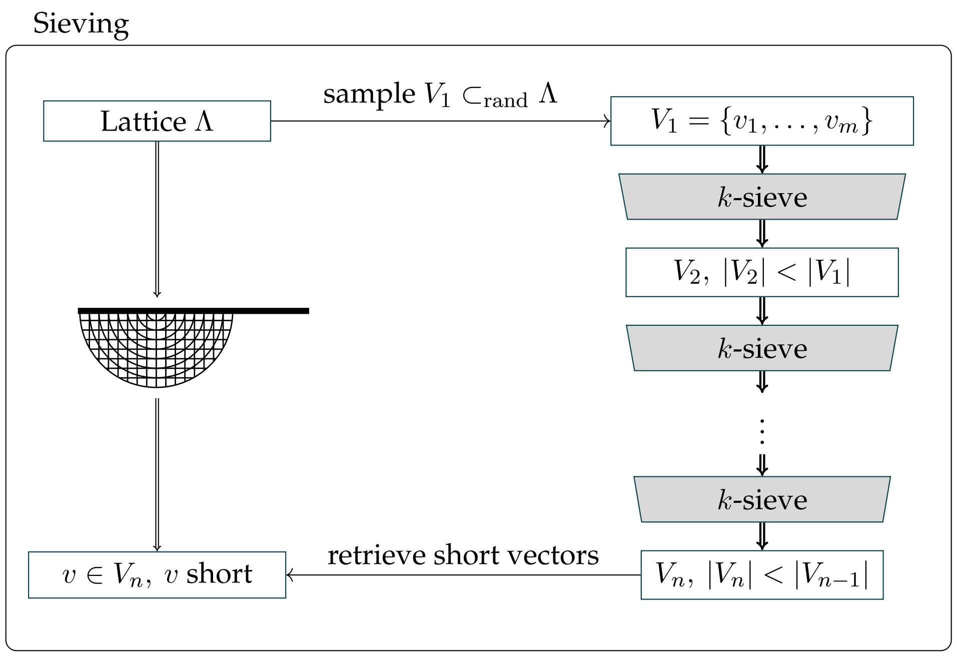

Figure 2.4: Sieving procedure, using a set of initial vectors, which are combined with a $k-$sieve, to create smaller sets of shorter vectors.

2.6.2 Enumeration

Enumeration proposed in [Poh81] and adapted as in [Kan83, FP85, SE91, MW15, ABF^{+}20] searches for a short vector in radius R by combining basis vectors. As there exists a vector v in the lattice Λ, which is arguably short in comparison to other vectors of Λ, there also exists a linear combination of the basis vectors of Λ, which results in v. To find this linear combination, one could search this combination with an exhaustive search.

Lattice enumeration algorithms exploit projections of vectors to find shorter vectors as our goal is to find a certain short vector. By searching a projection $\Pi_i(v)$ for the i-th basis vector with $||\Pi_i(v)||^2 \leq R^2$, a projection will never result in a longer vector. This projection is lifted to the next projection $\Pi_{i-1}(v)$ with the constraint $||\Pi_{i-1}(v)||^2 \leq R^2$. Thereby, the problem is abstracted to a linear problem of finding a scalar value u_i to shorten the resulting vector for the i-th lattice basis vector of Λ at a time. As there are multiple candidates for each scalar value, the general exhaustive search can be visualized as a tree search as shown in Figure 2.5. Every tree layer introduces the next scalar value, up to the leaf nodes, where the resulting vector v^*_{d-1} is a non-trivial linear combination of all basis vectors of Λ. The resulting vector is therefore relatively short as the projection constraints will reduce the length of the vector.

To enhance this procedure one can establish a pruning technique, which will cut off all

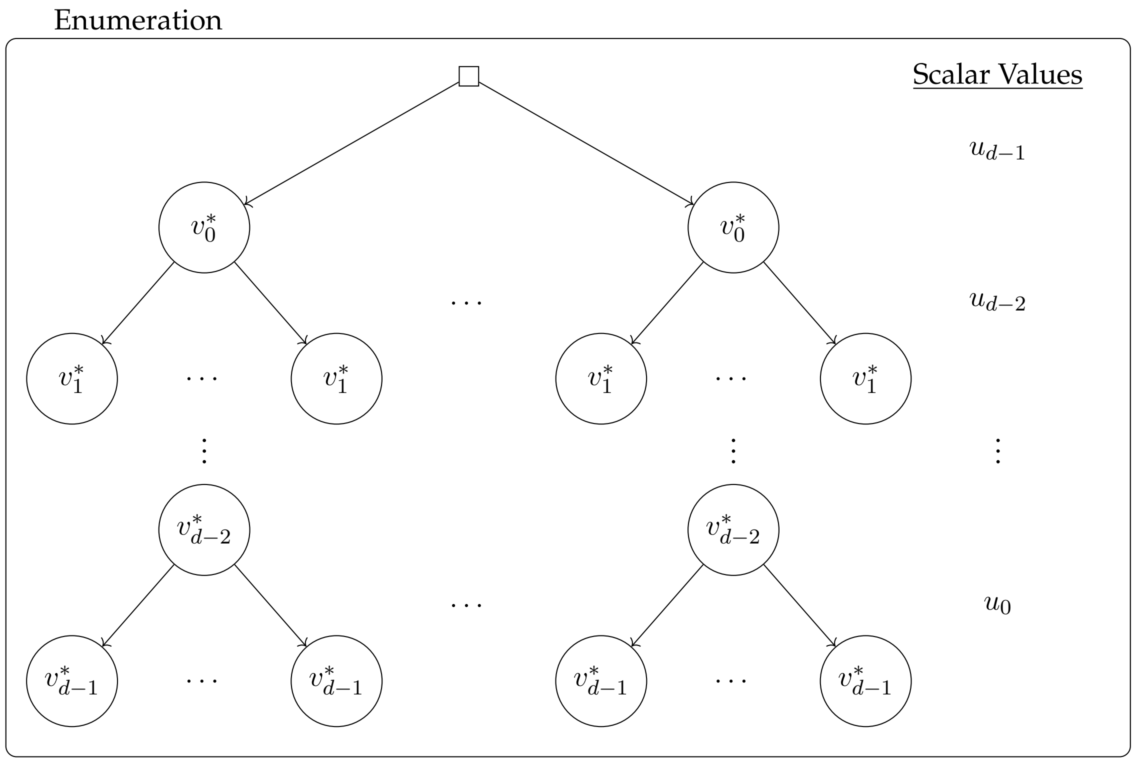

Figure 2.5: The general procedure of lattice enumeration. Utilizes a tree search, for finding the scalar values generating short vectors regarding a given radius R.

"irrelevant" subtrees. A subtree is considered "irrelevant" whenever the result does not fulfill the projection constraints. Thereby, a subtree is cut off if there is no scalar value for a candidate to fulfill the constraint. This pruning technique yields better performance for a "good" radius R. The radius R should be considered suitable if it is close to the norm of the shortest vector.

2.6.3 BKZ

Another approach to creating more performance-oriented solving algorithms for SVP is the BKZ algorithm [Sch87, SE91]. In general, the BKZ algorithm utilizes either lattice sieving or enumeration, which are used to solve a smaller block of the complete lattice basis of Λ.

The algorithm starts with the set of basis vectors $b_0, \ldots, b_\beta$ of Λ, where β is the block size parameter of the BKZ algorithm. For this block, the lattice sieving or enumeration is used to find projections for the given vectors. Next, the block is shifted, hence the next basis vectors are included in the block and the lattice sieving and enumeration are used again. This continues until the block reaches the last basis vector of Λ. The consequential projections of all basis vectors yield a new reduced basis, which is used to repeat the whole process. We call this loop a tour, where in general a small constant number of tours is enough to reduce the basis to a point where more tours would lead to small or no changes. Note that this technique is not polynomial in time complexity anymore.

2.7 Advanced Lattice Attacks

As lattice-based attacks became more popular, the lattice-based attacks discussed were optimized. On the one hand, computational power is less of a problem today than two decades ago, when the first lattice-based attacks on cryptographic systems were published. With more computational power one can break even harder instances with larger matrix dimensions. On the other hand, the "barrier" of those attacks seems to be unreasonable to break as lattice attacks often could not surpass certain parameters. Therefore, new attacks were developed to strengthen the attacks even more. One of which is [AH21], where their advancements not only strengthen lattice-based attacks on ECDSA but also broke the mentioned lattice barrier.

2.7.1 Lattice Attacks with Predicate

Albrecht and Heninger introduced a solution to the former "lattice barrier" in [AH21]. In many cases, lattice-based attacks could not surpass a certain amount of known bits for successful attacks and therefore literature designated a "lattice barrier". This barrier was believed to be unsurpassable by lattice-based attacks. Prior works such as LadderLeak ([ANT$^+$20]) showed that even partial knowledge of the most significant bit is enough to recover the private key d of ECDSA, whereas lattice-based attacks were bounded by needing more than four bits. By adding a predicate to the solving phase of the SVP, one can achieve better performance with fewer bits needed than previously thought. As [AH21] points out, by adding the non-linear knowledge of the ECDSA private key concerning the known public key, any solution found by solving the lattice problem can be checked for correctness. This enables the lattice-based attack to find a solution even if the target vector containing the solutions to the HNP is not the shortest vector of the lattice. All previous works are bounded by the fact that the attack is only possible if the shortest vector contains the solution to the HNP.
In their work, Albrecht and Heninger adapted existing implementations with a predicate version, which is the foundation for this work. They mainly focused on lattice sieving and lattice enumeration as these techniques can search through multiple vectors fulfilling given boundaries. These boundaries are given by the norm of the target vector, which in their setting can be larger than the shortest vector of the lattice. While simple approaches

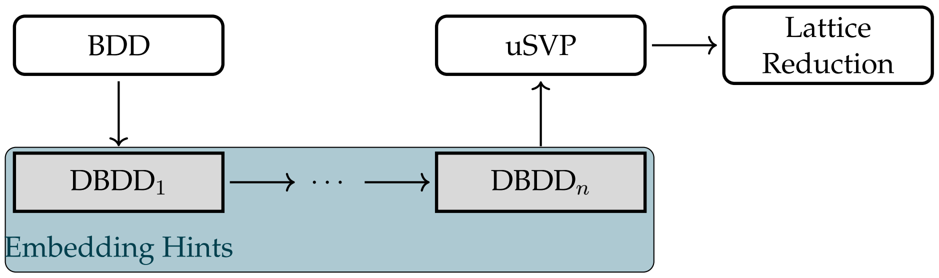

Figure 2.6: General procedure of lattice attacks using hints. For more detail see the original procedure of [DDGR20]. The reduced lattice problem Distorted Bounded Distance Decoding (DBDD), is a BDD problem, in which the smallest ball is changed to an ellipsoid.

such as the LLL algorithm can be used to find the shortest vector, they cannot search from this point on for larger vectors. Lattice sieving and enumeration on the other hand are search algorithms, which can find multiple candidates for a given target norm. By checking each candidate using the predicate, the correct solution can still be found.

2.7.2 Lattice Attacks with Hints

Another approach to strengthen lattice attacks is the adaptation of [DDGR20], which introduces "hints" as additional knowledge to be embedded into the lattice. In their work, the authors mentioned four types of hints: perfect hints, modular hints, approximate hints and short vector hints. Each of these types of hints provides some information about the lattice basis, whether it is the intersection with a hyperplane or the orthogonal projection of a vector v of the lattice. In general, this approach constructs a lattice by embedding the hints into a given lattice and solves the newly generated lattice with the usual techniques described earlier. As Figure 2.6 shows, the original problem is given as a BDD instance, which gives a lattice Λ. Assume, that one is given several hints $h_1, \ldots, h_n$, which are then embedded one after another into the lattice. The resulting lattice Λ_{h_n} is then transformed into an SVP lattice Λ', which can be solved using any lattice reduction presented above.
As stated in [AH21] their performance gains are higher than the performance gains given by including "hints". Furthermore, it is easier to create a predicate than to collect usable knowledge representable as hints because the predicate is often given directly by the protocol or further usage of exchanged data.

2.7.3 Lattice Attacks by Guessing Bits

Another approach to breaking the "lattice barrier", and generally improve the parameters, was published in [SETA22], where the authors guess more bits at the cost of solving exponentially many instances. Thereby they introduce multiple strategies for guessing bits: Guessing bits of the secret key and guessing bits of nonces. In both cases, one needs to solve an exponential number of lattice instances, from which only a few instances might result in the correct recovered nonces and thus the private key in case of an attack on ECDSA. By increasing the number of samples the method of guessing bits for the nonces has a non-negligible overhead. The overhead is exponential in the number of nonces and the number of bits guessed for each nonce. All these possible combinations need to be brute-forced to find the correct solution to the initial problem. This approach leads to a similar result as [AH21] by solving instances that could not be solved previously.

3 Lattice-based Attack with Predicate on DHKE

In this chapter we discuss, how to construct a lattice attack on DHKE as well as the general attacker model and our adaptions to the attacker model to obtain a predicate, which is used for advanced lattice-based attacks. Finally, we present our implementation which we built to evaluate the advanced lattice attacks on DHKE.

3.1 Lattice Attack on Diffie-Hellman

As discussed in Section 2.2, DHKE is used to share a secret between two parties. While this key exchange is widely used in many protocols, there exist also multiple attacks to break its security. By raising its security parameter to bit sizes of 3072, modern systems became secure against attackers as is. One attack on DHKE is a lattice-based attack on the HNP [BV96]. In this section, we discuss a lattice-based attack in its simplest and smallest dimension up to an upscaled and general version used for any known most significant bits. In its basic version, the lattice-based attack cannot be adapted to be used with the predicate version of [AH21]. We thus discuss a modified attack scenario to make it compatible. Although [AH21] stated that the predicate extension of lattice-based attacks can show similar results for any other lattice-based attack compatible with predicates, this result might vary for different setups. Therefore, the used framework of [AH21] was adapted to the presented predicate version of DHKE and the used modes of operation are presented in this section as well.

3.1.1 Lattice Attack on DHKE

As previously discussed in Section 2.1 on ECDSA, lattice attacks exploit side-channel information, which leads to short values in a crucial part of the protocol. Hence, we first discuss an attacker model applicable to DHKE.

Attack Model on DHKE (Lattice Attacks)

The attacker model for the following lattice attack on DHKE builds upon three crucial properties. The first of which is the reuse of one private value of one party (Bob) [MBA$^+$21], which can be represented by a server reusing its key b for multiple key exchanges. The second property is relatively trivial, as the attacker just needs to interact with Bob so that multiple exchanges between the attacker and Bob can be made. Finally, side-channel information for a crucial value of the protocol is needed. The value which should leak bits is the shared secret of which the first l bits must be known for the attack.

If the above-mentioned properties are guaranteed, the attack focuses on recovering the complete shared secret between Alice and Bob. The attack tries to recover the initial shared secret of Alice and Bob and also all shared secrets created by the attacker with Bob. As the attacker uses a public value, which is a combination of the public value of Alice and a randomly generated private value, he cannot calculate the shared secret himself. The additional shared secrets the attacker instantiated by using the public value as described have no other use than to recover the initial shared secret between Alice and Bob.

3.1.2 Construction of the Attack

In the following the construction of the lattice-based attack on DHKE is discussed. We explain, how to construct the lattice in the case of only a single sample, and extend this construction to a more general scenario.

Procedure for Simple Cases with one Sample ($m = 1$)

As Figure 3.1 shows, the scheme uses the public value of Alice to compute R_a and gain a resulting shared secret $s_r = g^{a \cdot b} \cdot g^{r \cdot b} = s \cdot g^{r \cdot b}$. The resulting shared secret is always a multiplication with the initial shared secret and thus one can use the lattice attack to solve for the multipliers. In general, this procedure can be repeated for multiple r values, which generates more samples for the lattice attack. The most significant l bits $k = \mathrm{MSB}_l(s_r)$ of the values s_r have to be known as well as the most significant values for s. We call the combination of the value r and k a "sample" as it yields additional information to break the initial shared secret s.

By collecting a sample as shown in Figure 3.1, one can construct equations similar to the ECDSA lattice-based attack. The following construction of the lattice base is based on [MH20]. As the shared secret s_r is a multiple of the initial shared secret s, one obtains the

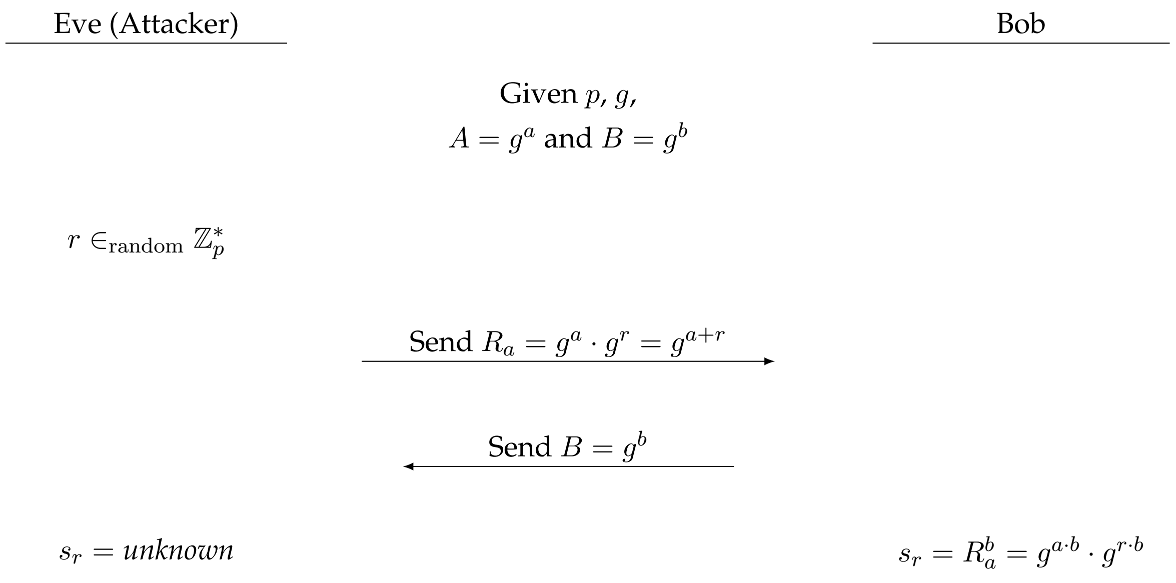
Eve (Attacker)
Bob
Given p, g,
$A = g^a$ and $B = g^b$
$r \in_{\text{random}} \mathbb{Z}_p^*$
Send $R_a = g^a \cdot g^r = g^{a+r}$
Send $B = g^b$
$s_r = unknown$
$s_r = R_a^b = g^{a \cdot b} \cdot g^{r \cdot b}$

Figure 3.1: Attack structure for lattice-based attack.

following equalities:

$$\begin{aligned}
k &= \mathrm{MSB}_l(s) \\
k_r &= \mathrm{MSB}_l(s_r) \\
t &= B^r \\
s &= k + q && (3.1) \\
s_r &= k_r + q_r \\
s \cdot t &= k_r + q_r \mod p && (3.2) \\
q - t^{-1} \cdot q_r + k - t^{-1} \cdot k_r &\equiv 0 \mod p && (3.3)
\end{aligned}$$

By combining Equation 3.1 and Equation 3.2, one can construct Equation 3.3. Relabeling Equation 3.3 with $u = k - t^{-1} \cdot k_r$ and $h = -t^{-1}$, one obtains a lattice to solve for the values q and q_r:

$$\Lambda_{m=1} = \begin{pmatrix} p & 0 & 0 \\ h & 1 & 0 \\ u & 0 & 2^L \end{pmatrix},$$

with $2^L > \max(\{q, q_r\})$. Thereby $\Lambda_{m=1}$ contains the vector $v_{sol} = (q, q_r, 2^L)$, which can be represented using the linear combination $v_{lc} = (q, q_r, 1)$ and $\Lambda_{m=1}$.

Procedure for General Case ($m \geq 1$)

As mentioned before, the procedure can be adapted to use multiple samples. The authors of [MH20] present an example with $m = 1$ and state, that it can be adapted to $m > 1$. In the following, we use this adaption and show that it does not directly work for $m > 1$, but needs to follow the construction we presented in Chapter 2 by constructing an HNP instance.

Adapting to the simple case, one needs to use multiple different values for r in the described attack scheme and collect the corresponding values k_i for all r_i used. With m of those samples one might think to adapt Equation 3.3 to obtain m equations with $i \in 1, \dots, m$:

$$\begin{aligned} & r_1, \dots, r_m \\ & k_0, \dots, k_m \\ & t_1, \dots, t_m \\ & s_0, \dots, s_m \\ t_i &= B^{r_i} \mod p \\ s_i &= k_i + q_i \\ q_0 - q_i \cdot t_i^{-1} + k_0 - k_i \cdot t_i^{-1} &\equiv 0 \mod p. \end{aligned} \tag{3.4}$$

Thereby s_0 is the shared secret of Alice and Bob. With the relabeling and the equations of the form of Equation 3.4, one can obtain a lattice matrix of dimension $m + 2 \times m + 2$:

$$\Lambda'_m = \begin{pmatrix} p & 0 & \dots & 0 & 0 & 0 \\ 0 & p & \dots & 0 & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & \dots & p & 0 & 0 \\ h_1 & h_2 & \dots & h_m & 1 & 0 \\ u_1 & u_2 & \dots & u_m & 0 & 2^L \end{pmatrix},$$

with $h_i = -t_i^{-1}$ and $u_i = k_0 - k_i \cdot t_i^{-1}$. At this point, this lattice is not suitable to recover q_0 as it contains the vector

$$v'_{\text{sol}} = \left(*, \dots, *, q_0, 2^L\right),$$

which is given by the linear combination of $v_{lc} = (*, \dots, *, q_0, 1)$. This solution vector might not be short, since every $*$ can be larger than 2^L even though it contains the value q_0, which is needed to recover the initial shared secret. The solution vector should look like

$$v_{\text{sol}} = \left(q_1, \ldots, q_m, q_0, 2^L\right)$$

as it contains every unknown value q_i and is relatively short due to $q_i < 2^L$.

To solve this problem and construct a new lattice containing v_{sol}, the equations and lattice matrix need to be adapted. By multiplying Equation 3.4 with t_i one can change the first m values of the target vector to solve for corresponding values of q_i instead of an undefined value $*$.

$$\begin{aligned} -q_i + q_0 \cdot t_i - k_i + k_0 \cdot t_i &\equiv 0 \bmod p \\ u_i' &= -k_i + k_0 \cdot t_i \\ h_i' &= t_i \end{aligned} \tag{3.5}$$

Note that the value t_i is multiplied by k_0 in Equation 3.5. Thereby the value k_i is not multiplied by any value, so the equation can be rearranged to $q_i \equiv q_0 \cdot t_i - k_i + k_0 \cdot t_i \bmod p$. With this construction, the target vector will include the values for q_i in addition to q_0. By updating the matrix of the lattice to this change with

$$\Lambda_m = \begin{pmatrix} p & 0 & \ldots & 0 & 0 & 0 \\ 0 & p & \ldots & 0 & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & \ldots & p & 0 & 0 \\ h_1' & h_2' & \ldots & h_m' & 1 & 0 \\ u_1' & u_2' & \ldots & u_m' & 0 & 2^L \end{pmatrix}$$

one finally can recover the solution vector $v_{\text{sol}} = (q_1, \ldots, q_m, q_0, 2^L)$. This vector can be constructed using the linear combination $v_{lc} = (x_1, \ldots, x_m, q_0, 1)$ with $x_i \in \mathbb{Z}$ and Λ_m and should be relatively short. By applying the algorithms discussed earlier, the SVP instance can be solved and all values q_i recovered.

Example

In the following an example is given to highlight the difference between both lattice structure and target vectors.

Given the above defined lattice basis Λ' and Λ, consider u_i and u_i', as well as the values h_i and h_i'. Choosing any $i \in \{1, \ldots, m\}$, one will get the following equations using the previously discussed solution vectors v'_{sol} and v_{sol}:

$$\begin{aligned} * \cdot p \pm q_0 \cdot t_i^{-1} \pm \left(k_0 - k_i \cdot t_i^{-1}\right) &\equiv 0 \mod p \qquad (3.6) \\ q_i \cdot p \pm q_0 \cdot t_i - k_i + k_0 \cdot t_i &\equiv 0 \mod p \qquad (3.7) \end{aligned}$$

Here Equation 3.6 was obtained by applying v'_{sol} with $\Lambda'_m\ (u_i, h_i)$, whereas Equation 3.7 was obtained with v_{sol} and $\Lambda_m\ (u'_i, h'_i)$. Note that in Equation 3.7 one fulfills this equation only if the i-th value of the solution vector contains q_i, as $q_0 \cdot t_i + k_0 \cdot t_i = s_0 \cdot t_i$ and can be rearranged using $q_i + k_i = s_i = s_0 \cdot t_i$ to $q_i = q_0 \cdot t_i + k_0 \cdot t_i - k_i$. Applying this procedure analogous to Equation 3.6, the i-th value of the solution vector will most certainly be a different group element than q_i. The i-th value $*$ of v'_{sol} needs to fulfill $* \equiv \pm q_0 \cdot t_i^{-1} \pm (k_0 - k_i \cdot t_i^{-1}) \mod p$.

Consideration of the Values in the Lattice Matrix

As the values in the lattice matrix Λ_m, might be unclear, each value is explained shortly. By inspecting the example above, one might see the dependencies of the values p, h_i and u_i. The example shows the dependencies of the values h_i and u_i as they cancel out if one finds the correct values for q_i and q_0. Furthermore, h_i and u_i are the known values of Equation 3.5. As the lattice cannot handle direct group arithmetic, the value p in each column adapts to this problem, since any multiplication with p converts values in $\mathbb{Z}_p^*$ to $\mathbb{Z}$. Another important fact about the values p in the diagonal sub-matrix of the whole lattice matrix is the linear independent filtering for the value of q_i for $i \in \{1, \ldots, m\}$. To find the value for q_0, the equation was built such that q_0 is combined with all values h_i and can be filtered by adding 1 to the second last row-vector. Finally, the value of 2^L is chosen to meet the norm of the solution vector, as we have $q_i < 2^L$ by assumption.
Note that the additional 1-bit extension discussed for ECDSA can be applied to Λ_m as well.

3.2 Lattice Attack on DHKE with Predicate

As seen in recent results regarding lattice attacks on ECDSA, introducing a predicate to the lattice attack gained better performance overall. Therefore, it is interesting to discover whether the discussed attack on DHKE shows similar behavior. A key difference between the attack on ECDSA and the attack on DHKE is the general lack of additional knowledge to check a solution candidate for correctness. As ECDSA recovers the secret signing key, the relation between the public and private keys can be used to check a recovered key for correctness. On the other hand, one only recovers the shared secret in DHKE, which

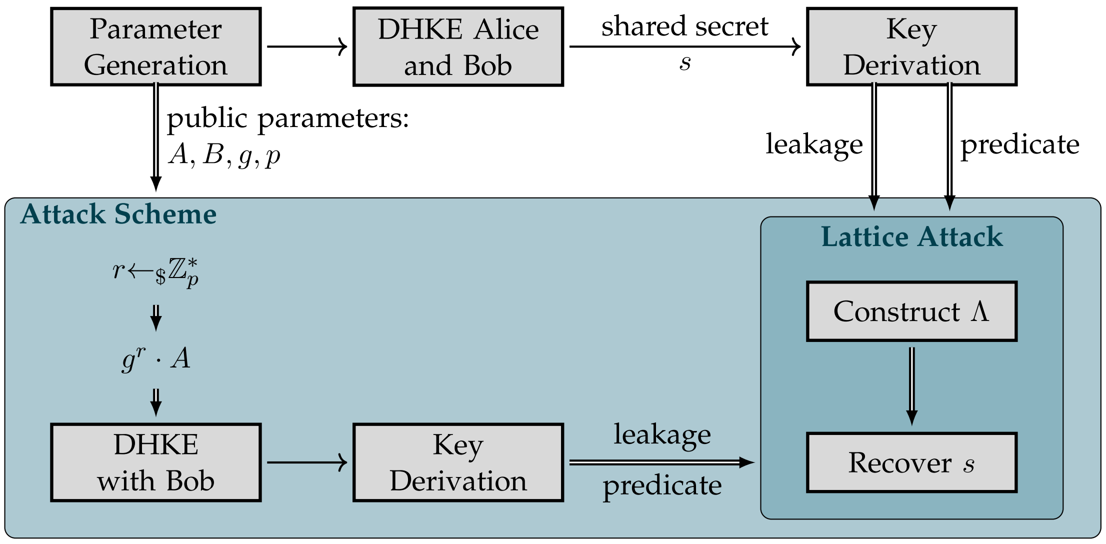

Figure 3.2: Lattice-based attack on DHKE with the predicate.

cannot be checked for correctness using any other given information of the protocol the attacker has access to. To check the correctness of the shared secret and to recompute the shared secret for the predicate, one private value is needed. As stated in the security description for DHKE (Section 2.2.1), an attack needs to compute the discrete logarithm to recover a secret value. Recovering a private value to check a recovered shared secret is not applicable as the predicate would solve the general problem by itself. Therefore, another strategy to check a solution candidate for correctness is needed. By widening the attacker model to include more usable knowledge for the predicate, a reasonable predicate can be formed, which is described in more detail in the next section.

3.2.1 Attacker Model

The new attacker model adapts the model described in Section 3.1.1. Additionally to the described model, the shared secret is used in further algorithms, which are also known to the attacker. In Figure 3.2 we give a brief overview of which parts of the DHKE protocol give information to the attacker and how the attacker adapts it to create a lattice-based attack. In general, the shared secret is used to derive an encryption key to encrypt certain data m. As protocols in general need to agree on algorithms used later on, the attacker also knows the used encryption and key derivation algorithm.

This means that the attacker recovered the most significant bits of the shared secret s as well as a ciphertext c, the key derivation algorithm H, the encryption algorithm $\mathrm{Enc}(m, k)$ and the decryption algorithm $\mathrm{Dec}(c, k)$. The message m is encrypted to ciphertext c using shared secret s as follows:

$$k = H(s) \tag{3.8}$$

$$c = \mathrm{Enc}(m, k) \tag{3.9}$$

In modern protocols such as TLS, the messages sent between two parties are encrypted using a block cipher such as AES, which encrypts and decrypts the message symmetrically. As both parties need the same symmetric key, they first need to come to an agreement on one. To solve this problem a key exchange is used, which in our case is handled using DHKE. As the shared secret might be too large for the symmetric encryption key, it is often combined with a key derivation function to generate a key of suitable length for the encryption algorithm.
Protocols like TLS use a combination of symmetric encryption algorithms with a key derivation function. Therefore our new attack model should be compatible with many real-world applications, as one just needs to set the correct algorithms. Concrete algorithms used for our testing purposes will be described in Section 3.3.

3.2.2 Leakage

We already discussed how leakage is obtained in the ECDSA scheme. By adapting the same methods, one could leak information about the private values a and b, as the computation of g^a uses a similar procedure, compared to the successive point addition in ECDSA. While this seems to be applicable for the described lattice attack on DHKE, it is not, as the attacker model described in Section 3.1.1 uses leakage of the shared secret and not the private value of any party. Therefore we cannot use leakage obtainable from the core protocol but need to leak information of s through further usage of the shared secret. In [MBA$^+$21] the authors found a side-channel attack on TLS, which leaks the most significant bits of the shared secret by measuring the timing behavior of the further used key derivation algorithm. Figure 3.2 shows that the leakage needed for the presented lattice attack does not come from the core protocol but through the used key derivation. This is one central key difference to the lattice-based attack on ECDSA, as their leakage can be received by signing data, which is the core algorithm.

3.2.3 Predicate

The new attacker model allows the attacker to check a shared secret for correctness, as it can be used to decrypt the captured ciphertext c. If the attack can decrypt the ciphertext successfully by using the shared secret and the appropriate key derivation function, the

shared secret is recovered correctly. The probability of finding a shared secret that results in a collision of the key derivation function and therefore decrypts the ciphertext successfully is very small, as the key derivation function is built to minimize the probability of two different inputs resulting in the same output.
To decide whether the decryption was successful, there are two possibilities. One is that the decryption algorithm supports integrity checks of the message, causing the decryption to fail with an error, whenever the ciphertext is not decrypted with the correct key. The second solution is less concrete, as in this approach the decryption will not fail using a wrong key, but rather output an arbitrary byte sequence. In this case, one has to check whether the byte sequence is the correct plaintext, for example, readable text. As many algorithms encrypt a specific header, which is a certain publicly known byte sequence at the beginning of the message, it can be checked whether the correct header was decrypted. This leads to a performance advantage, as it is sufficient to decrypt the first blocks of the ciphertext to check for the header bytes. In most cases, block ciphers will decrypt all successive blocks if one has the correct key and initialization data the first block. As encryption and decryption using AES are highly optimized in modern hardware, the performance difference is not as concerning as one might think. Additionally, larger bit-sizes for DHKE, for example, 3072-bit, are more costly than the optimized AES instructions. To decrypt only a constant number of blocks of the ciphertext guarantees only an advantage in performance with some algorithms and if the ciphertext is arguably large in comparison to the constant number of blocks for the encrypted header.

3.2.4 Modes of Operation

With a predicate suitable for DHKE, its performance gain by including the predicate can finally be measured. As the framework of [AH21] used different methods to adapt the lattice attack with the predicate, we discuss the different algorithms used in the framework and describe the different modes, which we will use for the DHKE adaptation as well.
To compare the addition of the predicate, we use the standard implementation of LLL of SageMath [The22] as a baseline and test it against every operation mode available in the framework of [AH21].

LLL

The SageMath framework supports a wide range of cryptographic algorithms. It also supports state-of-the-art implementations for lattice reduction such as the LLL algorithm. Therefore, this implementation is used to create a performance baseline in terms of parameter sizes and time.

Enumeration

The framework created in [AH21] supplies multiple usable algorithms for lattice reduction with predicates. As we previously discussed, lattice enumeration and sieving are key algorithms used for lattice reduction. Therefore, the framework implements both techniques with a predicate adaptation. In the case of lattice enumeration with a predicate, previous implementations may be adapted with predicates relatively easily. As the enumeration finds vectors within a given radius R, those vectors are candidates to solve the HNP instance. If a candidate is found, it can be checked with the defined predicate. Otherwise, the exhaustive search will be continued to find new candidates.

Sieving

The sieving implementation also provides the opportunity to include a predicate in the procedure. In the sieving procedure, sets of vectors are reduced using k-sieves to create a new set of shorter vectors. If one of the new vectors is at any point within radius R, one can additionally check whether this vector is a possible solution to our lattice problem by testing with the predicate. Therefore, sieving, as well as enumeration, are used for our testing, as they both can be adapted with a predicate.

Enumeration and Sieving with BKZ

Finally, the framework of [AH21] supports an additional mode for sieving and enumeration, where these modes are combined with the BKZ algorithm. In this case, a BKZ reduced basis can be checked for containing short vectors in the given bound. If a short vector is found, it is further checked using the predicate.

Automated Selection

The last mode used for evaluation and implemented by the framework is an automated selection. This selection consists of the enumeration and sieving technique as well as their combination with BKZ. By some given boundaries for the target norm and Gaussian Heuristic of the lattice, this mode decides which technique to use. It is also the default mode for the framework which is another reason why we included this mode.

3.3 Implementation

As discussed above, our implementation utilizes the framework of [AH21] as well as all described modes of operation: enumeration and sieving with predicate as well as enumeration and sieving using BKZ and the predicate. In general, our implementation is heavily

inspired by the structure of the ECDSA version of [AH21], as it includes the generation of samples and instances as well as benchmarking and solving functions.
Thereby, our implementation is capable of generating instances as well as solving them. It also supports benchmarking modes for a set of instances to compare different solving algorithms on the same inputs. In the following, the generation of samples and parameters for the generation and running of the attack are described in more detail.

Sample Generation

As for extensive testing, multiple test instances with a large set of different parameters are needed. Our implementation supplies a generation of instances. The DHKE lattice-based attack recovers a shared secret by collecting samples. Therefore, the implementation can generate an instance with a given number of samples to a previously generated initial shared secret. First, one needs to set the bit size used to generate a safe prime number. As the generator for multiplicative groups is often set to 2, our implementation uses this value as default. Note that we can set the generator to 2 because we use safe prime numbers. The generated instance can be stored in a file to be reused for different testing setups. At this point all values, public and private, are stored, so the instance can be used for multiple parameters (for example the number of leaked bits).

Generation Parameters

In general, our implementation supports multiple settings for the main parameters. First, the bit size of the DHKE protocol can be set, which influences the resulting prime number. One has to either set a static prime number matching the bit size or let our implementation generate a safe prime number with an appropriate bit size. Next, the number of samples m needs to be set, which generates m random private values $r_1, \ldots, r_m$ to calculate the shared secrets regarding the attack scheme of Figure 3.1. By generating the samples the shared secrets are used to encrypt a default message M, which can be changed. The encrypted message later is used to evaluate the predicate.
As the bit size might vary from benchmark to benchmark, the shared secrets are not used directly for encryption. We follow the principle of the attacker model described in Section 3.2.1. Therefore, we utilize a hash function to get a constant key length for encryption and decryption. Our framework uses the SHA256 hash function, as it should be sufficient for the generated instances. For encryption, we use AES in CBC mode, as it is a common mode of operation. To keep the structure simple, the plaintext used for encryption stays the same, as it should make no difference in leakage because each plaintext is encrypted using a different key regarding the hash function. Note that the aforementioned imple-

mentation details could be exchanged easily for any other hash and encryption algorithms as well as different plaintexts and ciphertexts to be adaptable to other distinct real-world attacks. The attack assumes that hashing and encryption algorithms are computationally secure and not the weakest link of the attacked protocol.

Running an Attack

To run an attack an instance is used with multiple extra parameters, providing the number of used samples and the number of leaked most significant bits of each shared secret. Using default modes, the implementation forms the lattice matrix out of all given values and performs the selected attack as described in Section 3.1.1. The implementation supports all described modes of operation as the framework of [AH21] as well as the LLL mode as a reference implementation.

4 Evaluation

In this section, we evaluate lattice attacks on DHKE using predicates. Thereby we are using the LLL integration of SageMath as well as the previously discussed framework of [AH21].
The SageMath framework provides fast implementations for many cryptographic calculations. Thus, the included LLL algorithm is highly optimized. We use our implementation to create instances to benchmark the different techniques and modes of operations and to examine if the results of previous works attacking ECDSA are applicable to DHKE as well.
We tested multiple parameters of the lattice attack to display the relation between lattice dimension (related to the number of samples), the number of known most significant bits, time complexity and bit size of the protocol.

4.1 Sample MSB Correlation

In this section, we evaluate the correlation between the number of samples and the number of most significant bits to attack DHKE. Accordingly, we discuss different settings for the overall bit-size of the DHKE protocol: 128, 256 and 512-bit. While commonly used bit-sizes for DHKE are 2048-bit and 3072-bit, we focused on lower bit-sizes to test for larger numbers of samples and to find the relation between the different bit-sizes, which also indicate the performance for larger bit-sizes. Additionally, there are problems with the framework of [AH21], because their implementation seems to have issues with bit-sizes above 521-bit, which is the maximum number of bits for ECDSA they tested with. When using the framework with larger bit-sizes, the solving algorithm terminates with errors. We assume that this issue is fixable, but it was out of the scope of this work to correct this issue. For evaluation, we will first explain the expectations, which are based on [AH21], and then discuss our strategy to benchmark the different settings.

4.1.1 Expectations

Our expectations are formed based on the results from [AH21] as they were able to attack ECDSA instances with fewer bits by introducing the predicate. The authors proposed that their technique should apply to all kinds of lattice attacks with usable predicates. We discussed our new attempt to introduce a predicate to the DHKE lattice attack, which should be applicable, as its performance overhead is negligible. Therefore, we should observe that the techniques such as enumeration and sieving combined with the predicate will solve parameters that were not solvable with standard techniques like LLL.

4.1.2 Strategy

To find the correlation between the number of samples and leaked bits, we need to test different settings for these parameters. When comparing these different settings, one discovers that more samples are needed when there are fewer known bits. For this reason, we first generate an instance with a high number of samples, so that the number of samples can be increased gradually.
Algorithm 2 shows how we find the minimum number of samples needed to recover the shared secret for a given number of leaked bits. It will try to solve each number of leaked bits in the range of $\mathrm{min}_{\mathrm{msbs}}$ to $\mathrm{max}_{\mathrm{msbs}}$. The function Solve uses the specified algorithm (ALG, which is one of the described modes of operation from Section 3.2.4), the generated instance, number of leaked bits and number of samples. Algorithm 2 increases the number of samples up to the point where the instance can be solved. Note that the algorithm starts with a higher number of leaked bits, which is decreased, while the number of samples is increased. This helps to reduce the time needed to benchmark the settings. It is very unlikely that an instance with fewer leaked bits is solvable with fewer samples than the same setting with more leaked bits. Hence, we imply that the number of samples is more likely to increase with fewer leaked bits. If one is given more samples than the minimum discovered by our algorithm, the instance should be solvable, as more samples will only decrease the norm of the shortest vector relative to the lattice.

4.1.3 Results

Hereafter, Algorithm 2 is used to benchmark the different bit-sizes starting with 128-bit. The following plots compare the different techniques discussed in Section 3.2.4.

Bit-Size of 128-bit

Figure 4.1 shows that the predicate techniques perform slightly better than the LLL algorithm. The expectation discussed above implies that standard techniques such as LLL meet a barrier at a certain point, which can be surpassed using the predicate approach. Evaluating the range from 3 leaked bits to 6 leaked bits, one can see that the number of required samples increases faster when using the LLL algorithm compared to the predicate techniques. The predicate techniques `enum_pred` and `sive_pred`, which are the predicate techniques without BKZ, show the best performance. They may even solve instances

Algorithm 2: Sample-MSB-Correlation Benchmark

```
Input: Algorithm $\mathrm{ALG}$, number of samples: ($\max_{\mathrm{samp}}$, $\min_{\mathrm{samp}}$) and number of
       leaked bits: ($\max_{\mathrm{msbs}}$, $\min_{\mathrm{msbs}}$).
Output: Benchmark results, i.e. mapping of least amount of samples needed to
        recover a secret for a given number of leaked bits.
1  $I = \mathrm{GenerateInstance}(\max_{\mathrm{samp}})$
2  $\mathrm{samples} = \min_{\mathrm{samp}}$
3  for $i \in \{\max_{\mathrm{msbs}}, \dots, \min_{\mathrm{msbs}}\}$ do
4      do
5          $\mathrm{solved}, \mathrm{solution} = \mathrm{Solve}(I, \mathrm{ALG}, \mathrm{samples}, i)$
6          if not solved then
7              $\mathrm{samples} = \mathrm{samples} + 1$
8          else
9              $\mathrm{AddResult}(\mathrm{samples}, i, \mathrm{solution})$
10     while $\mathrm{samples} \leq \max_{\mathrm{samp}}$ and not solved
```

with 3 known bits, which is not possible for the other approaches. Another observation is the reduced number of samples needed, as `enum_pred` and `sive_pred` could solve instances with the same number of samples but fewer known bits compared to the LLL algorithm or the other techniques of the framework of [AH21].

Note that all techniques grow exponentially in the number of samples. This also affects the time needed to solve an instance, as the matrix dimension is related to this number of samples. A Larger matrix dimensions always results in more computational effort. Thus, one can expect similar behavior when correlating computational time and the number of samples.

Bit-Size of 256-bit

In the following, we review the same experiment but with a bit-size of 256-bit. With a higher bit-size there are two main properties: One is that the calculations on the values are getting more costly as all values generally double in size for 256-bit compared to 128-bit. This property is further evaluated in Section 4.2.2. The second property is the number of bits needed to successfully recover secrets. Our results show that the higher bit-size increases the minimum number of most significant bits needed. This can also be seen when comparing Figure 4.2 with Figure 4.1. Figure 4.2 shows that about 60 samples are needed to solve an instance with 6 leaked most significant bits. On the other hand, about 60 samples can solve instances with only 3 to 4 bits leaked on 128-bit.

In general, the behavior of the techniques seems to be similar to 128-bit, as the same predicate techniques outperform the LLL algorithm and other variants. The techniques

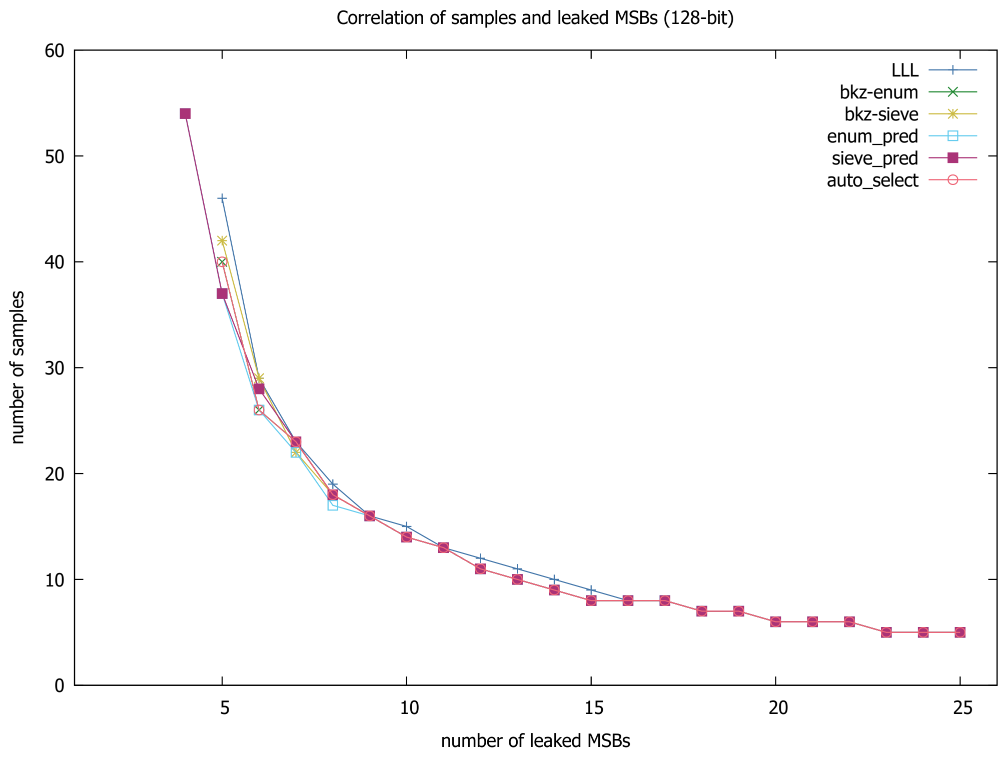

Figure 4.1: Resulting correlation between the number of samples and the number of known most significant bits. The DHKE scheme used for this plot uses a 128-bit prime number and the generator $g = 2$.

`enum_pred` and `sieve_pred` both show that the predicate addition helps to solve instances with fewer samples than the LLL algorithm and the other variants. But this enhancement comes with a higher cost in time complexity.

Bit-Size of 512-bit

Finally, the bit-size is set to 512-bit. As this is the highest bit-size used for this evaluation, it is the bit-size that is nearest to commonly used bit-sizes for DHKE. In Figure 4.3, one can see that the expectations are met as well because the exponential growth of `enum_pred` and `sive_pred` is lower than the exponential growth of the LLL algorithm.

We displayed all three bit-sizes to show the correlation between the number of samples and the number of known most significant bits. At this point, one can see that more samples are needed to solve instances with the same number of leaked bits when the bit-size increases. Using a constant amount of samples one can also expect to double the

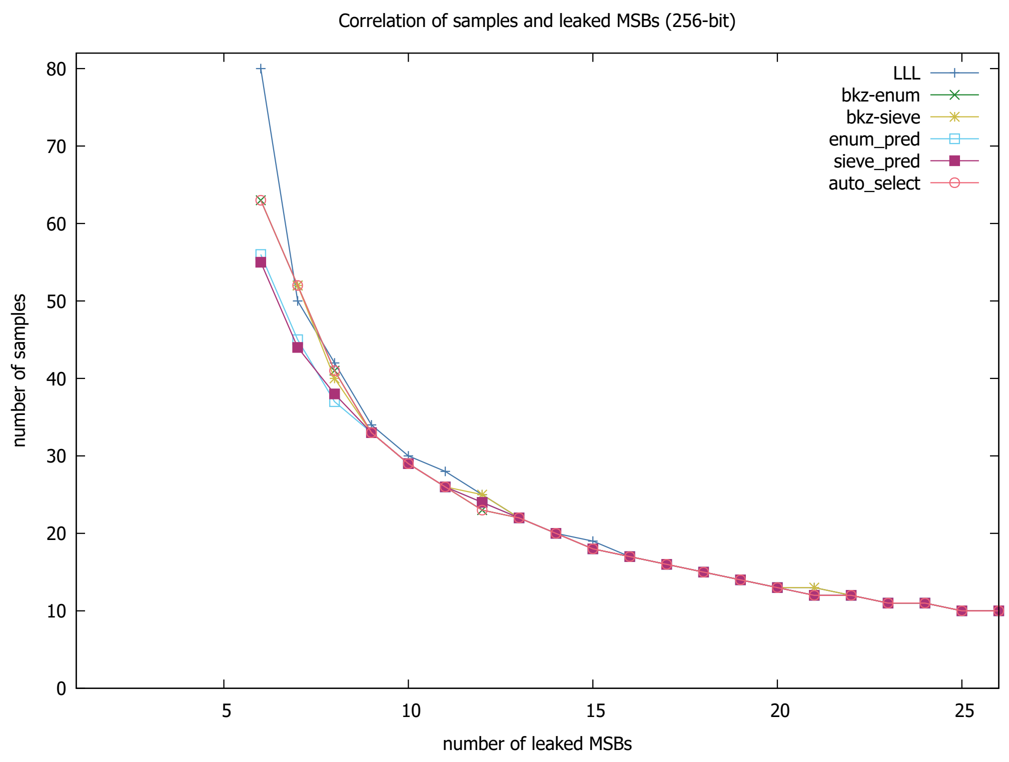

Figure 4.2: Resulting correlation between the number of samples and the number of known most significant bits. The DHKE scheme used for this plot uses a 256-bit prime number and the generator $g = 2$.

number of leaked bits to recover a shared secret if the bit size is twice as large.

4.2 Time Complexity

In Section 4.1 we discovered that the number of samples needed increases exponentially with fewer bits. Increasing the number of samples also directly influences the matrix dimension of the lattice, which is to be reduced. Lattice reduction algorithms typically scale with the dimension of the matrix. Therefore, the number of samples has a high impact on the practicality of the lattice attack. In this section, we compare the time needed by the different evaluated techniques with each other as well as the differences for higher bit-sizes. The evaluation is based around Figure 4.4, which includes the time measurements we made for the different bit-sizes, as well as the differences between the techniques.

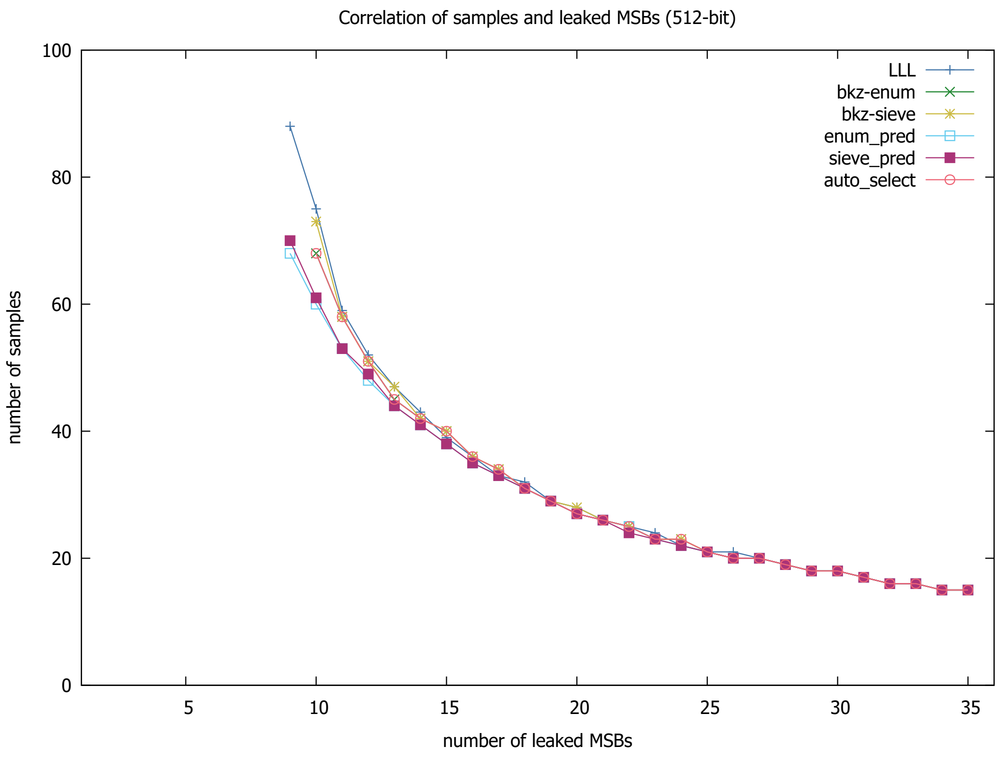

Figure 4.3: Resulting correlation between the number of samples and the number of known most significant bits. The DHKE scheme used for this plot uses a 512-bit prime number and the generator $g = 2$.

4.2.1 Time Complexity of Lattice Reduction Techniques

As was previously discussed lattice reduction techniques are dependent on the matrix dimension, which is related to the number of samples. The plots in Figure 4.4 shows the number of samples concerning the time in seconds needed for the reduction. They range from 10 to 44 samples but differ in the time needed as they scale differently with the number of samples. To measure the time, the current time is taken before the reduction algorithm is started and after the algorithm is finished. This provides the needed time for running the algorithm.

Beginning with the LLL algorithm one can see a non-linear tendency for a bit-size of 512-bit. The lower bit-sizes seem to behave generally linear up to 35 samples. Comparing this behavior to `bkz-enum` and `bkz-sieve`, one can see that the BKZ techniques have a linear tendency as well as a higher gradient. When compared to LLL, sieving and enumeration (`bkz-enum` and `bkz-sieve`) need more time with the same number of samples. On the other hand, both `enum_pred` as well as `sieve_pred` show a non-linear trend for even the lower bit-sizes. Consequently, the gradients of `bkz-enum` and `bkz-sieve`

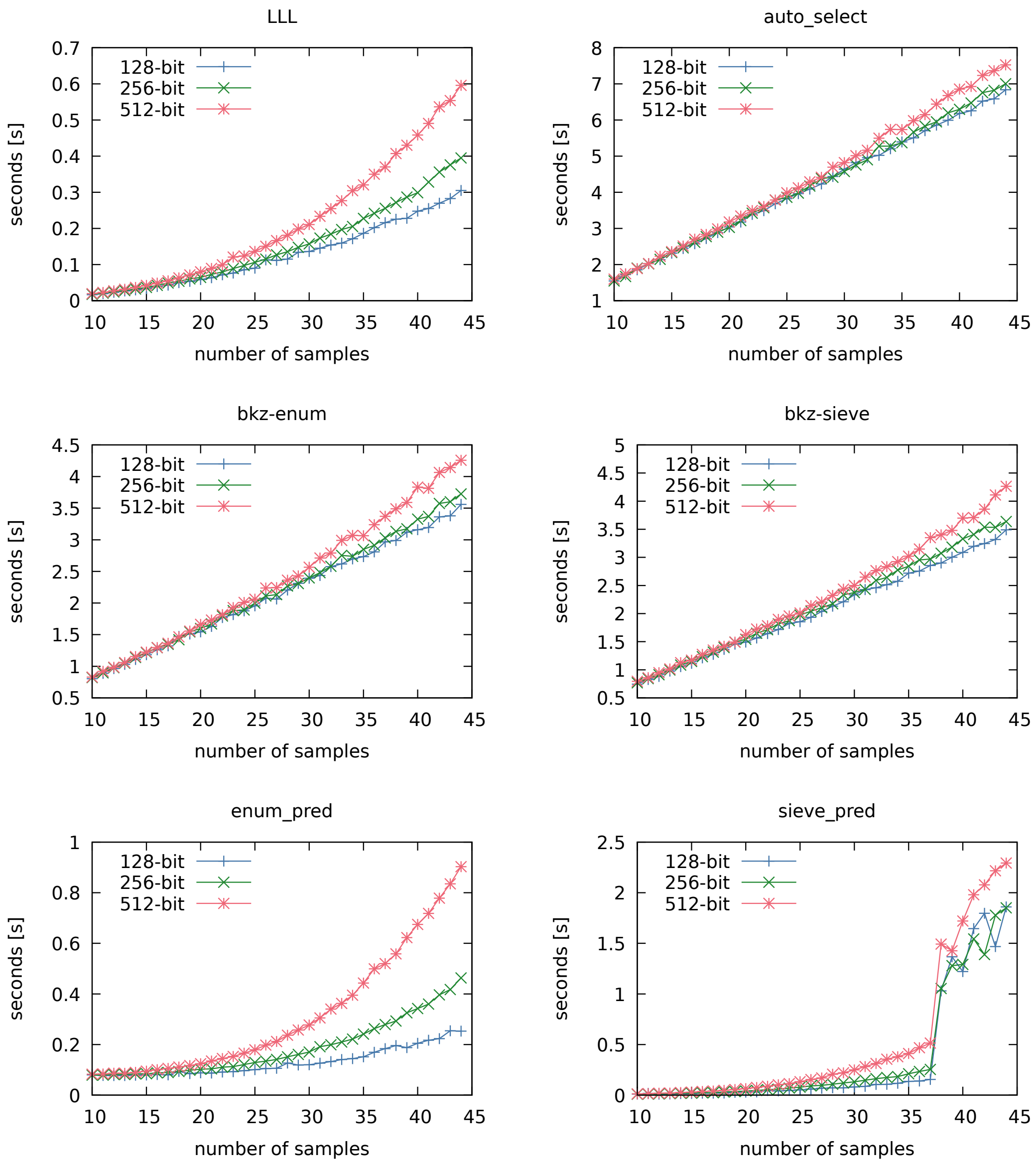

Figure 4.4: Time to number of sample relation of each technique.

barely change and are surpassed by `enum_pred` and `sieve_pred` at a higher number of samples as both enumeration and sieving grow in a non-linear way. Note that the `sieve_pred` technique shows a consistent time behavior up to 37 samples. After this point, the growth appears to be shifted by a constant amount. In general, this technique had the most outliers when tested, which might be since sieving is dependent on randomness. Lastly, the technique `auto_select` seems to be the slowest for a small number of samples. But, as already stated for the BKZ techniques, `auto_select` runs faster

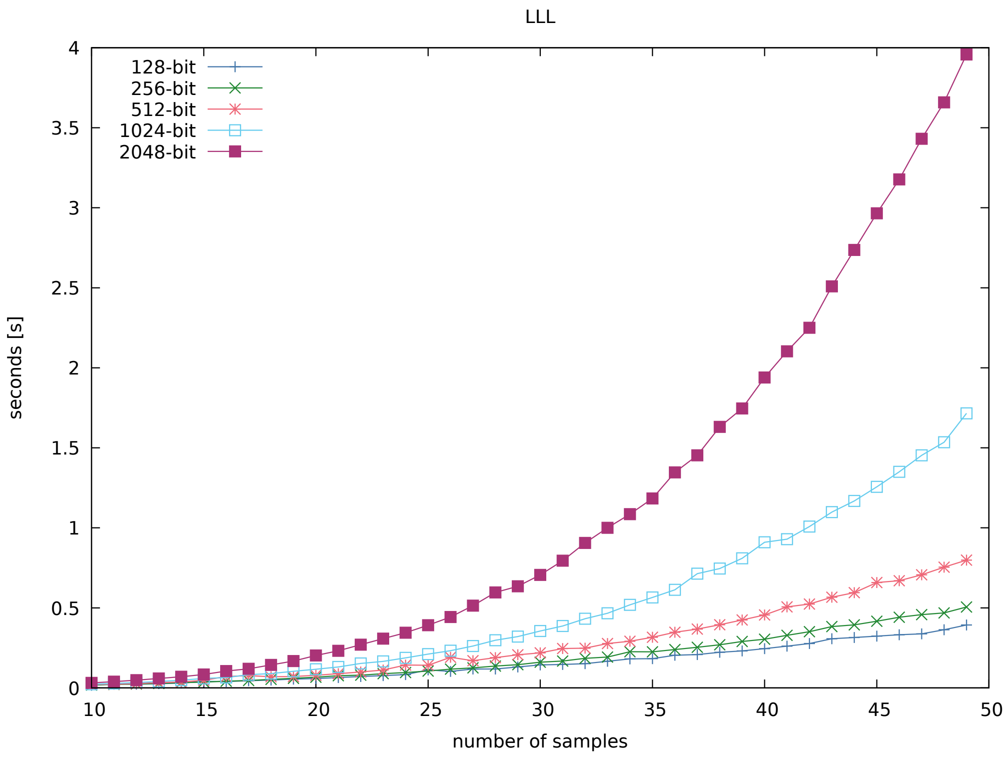

Figure 4.5: Timing behavior for the LLL algorithm for different bit-sizes.

on larger numbers of samples than `enum_pred` and `sieve_pred`, as the computational time rises linearly. This linear tendency does at least hold for the parameter sizes we tested. Note that `auto_select` almost always selects the BKZ techniques, which result in similar running times.

As the LLL algorithm runtime seems to grow exponentially in the number of samples, we tested larger numbers of samples and even larger bit-size in Figure 4.5. When comparing this to the other plots, one can clearly see that the execution time grows exponentially, especially for larger bit-sizes.

4.2.2 Time Complexity with Different Bit-Sizes

Previously, we evaluated the time behavior of the different techniques. But as DHKE is commonly used with larger bit sizes, it is necessary to investigate the impact of the bit size on the running time. Therefore we compare the bit-sizes of 128 to 512 for each technique.

While one can see that an increase in bit-size almost always results in more time needed, it differentiates between the techniques. Whereas the impact of larger bit-sizes seems to be constant in the BKZ techniques, the behavior for LLL, `enum_pred` and `sieve_pred`

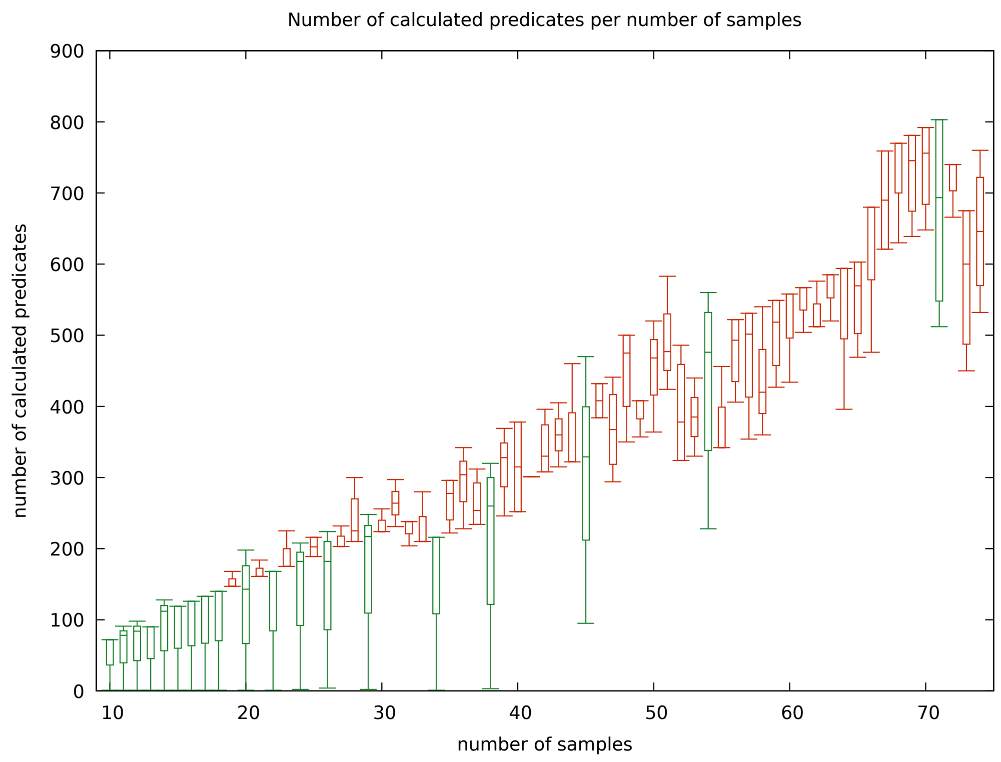

Figure 4.6: The Number of computed predicates to the number of samples for `enum_pred` with a bit-size of 256-bit. The green box plots included at least one successfully recovered shared secret, while the red box plots consist of only failed recoveries.

differs, since they seem to be impacted at least by a linear factor.

To more precisely evaluate this tendency Figure 4.5 shows larger bit-sizes. Here, one can see that the exponential growth continues for larger bit sizes. For our tested range, it appears that the impact of the bit-size is at most linear.

4.2.3 Effect of the Predicate on Time Complexity

The advanced techniques utilizing the predicate have another factor that could potentially increase their running time: the time complexity of evaluating a predicate. We previously discussed that our procedure performs relatively well. One could even reduce the computational overhead to a constant whenever executing an attack with partial knowledge about the plain text that involves a constant number of encrypted blocks. It was also mentioned that theoretically this overhead can be reduced using efficient AES instructions. Therefore the overhead can be considered constant.

Figure 4.6 shows the relation of the number of computed predicates for an instance with

certain numbers of samples. In this figure, we include both the results of failed attempts (red) of solving a specific instance as well as instances that could be solved (green). The results shown in the figure resemble a run of Algorithm 2, but with the adaption of also collecting the results from failed solving attempts. One can see, for up to 38 samples some instances finished successfully while using the predicate only once. This is because the predicate techniques use a form of standard lattice reduction and start with an already reduced basis, also visible when comparing Figure 4.1 to Figure 4.3. In these figures, all techniques perform similarly to each other up to a certain point. Up to this point, the techniques using the predicate will recover the correct shared secret without even using advanced techniques such as sieving and enumeration. When inspecting Figure 4.6, one can see that no instance could be solved using the predicate only once if the number of samples is above 38. The instance with more than 38 samples was only solvable using the predicate multiple times. Another observation is the linearity of the number of used predicates. While the maximum number of computed predicates rises linearly, the minimum number of calculated samples rises linearly with a smaller gradient. Therefore the expected range of how often the predicate is verified increases.
Our results conclude that the number of predicates grows in a linear fashion with the number of samples, whereas the computation time grows exponentially in the number of samples. Therefore even computationally expensive predicates are feasible for this lattice attack adaptation.

5 Discussion

In this chapter, we scrutinize the practicality of the DHKE lattice attack and discuss mitigations to counteract these attacks. Finally, we consider possible advancements to the evaluated attack as well as conspicuous features that we have noticed.

5.1 Practicality of the Results

Our evaluation shows that the general concept of lattice attacks with predicates applies to DHKE. The improvements of the attacks show similar results as in previous works on ECDSA [AH21], as parameters out of scope for previous techniques could be solved. That said, there is a significant problem with the efficiency of the lattice attack on DHKE, which comes into place for larger bit-sizes.
The number of samples needed to solve an instance with a constant number of leaked bits increases with the bit-size. Therefore, in commonly used settings of bit-sizes above 2048-bit, the number of leaked bits that are necessary to successfully solve the secret is very large. For small bit-sizes we can break parameters with 3-4 leaked bits per sample and large amounts of collected data. When considering larger bit-sizes, one would easily need about 20 to 40-bits leaked per sample, which should be difficult to obtain in most cases by simply collecting enough data points. The attack becomes practically interesting if these conditions are met.
The second problem is the increased number of samples one needs for fewer bits leaked. This directly affects the time performance, which increases exponentially as well. To break instances with a high number of samples, large data sets would be needed to supply enough samples as well as computationally powerful systems to handle lattices of larger dimensions. The data collection in some cases of lattice-based attacks in ECDSA such as TPM-Fail [MSEH20] exceeded more than 40000 handshakes. If one compares that to DHKE in a realistic scenario, where a leakage of 20 most significant bits is needed, a much higher amount of key exchanges is needed compared to the number of handshakes in the case of ECDSA. We discussed that leakage in the case of DHKE often results in leaking the most significant zero bits. Therefore, one needs to make at least 2^{20} key exchanges, to increase the chance that at least one exchange results in a shared secret with 20 leading zero bit. Even if the side-channel has a perfect leakage of the 20 most significant bits, one only obtains a single sample by 2^{20} key exchanges. The overall number of key exchanges nec-

essary for successfully launching the described attack is relatively high when compared to similar attacks on ECDSA as mentioned before.
Considering the described problem, the practicality of this lattice attack seems to be limited. But while this lattice attack has its disadvantages, many implementations and servers still fulfill some of the needed properties, such as the reuse of private values [MBA+21].

5.2 Mitigations

Lattice-based attacks on any cryptographic system are a threat. In the case of ECDSA and DHKE, these attacks are practical if the leakage is high enough. Without any leakage, they could be ignored, since up to now no lattice-based attack exists that does not rely on leakage. But, as hardware and implementations become more complex, the underlying cryptographic systems are more vulnerable to side-channel attacks. We discussed the existence of multiple side-channel attacks, which are capable of collecting enough information through given leakage to recover signing keys or shared secrets. In the following section, we discuss potential mitigations and additions to the DHKE.

5.2.1 DHKE with Elliptic Curves

We discuss in Section 2.2 an alternative key exchange technique called ECDH. By using ECDH instead of DHKE, one could mitigate the attack we evaluate in this work. The DHKE attack exploits the fact that one can combine one party's public value with their private value, which results in finishing the key exchange in a multiplication of $g^{r \cdot b}$ and $g^{a \cdot b}$. By turning the multiplication into an HNP instance, one can solve this multiplication for $g^{a \cdot b}$. When comparing DHKE to ECDH, one will observe that we can completely follow the attack scheme described in Section 3.1.1. The key difference between DHKE and ECDH is the group it operates on. Combining $g^{r \cdot b}$ and $g^{a \cdot b}$ in the ECDH scheme is not a multiplication in $\mathbb{Z}_p^*$ but an addition between two elliptic curve points, as $g^{a \cdot b}$ relates to point $P = (a + b) \cdot G$ with generator point G. Point addition on elliptic curves cannot be rearranged to a multiplication solvable by a lattice-based attack as far as we know. Comparing it to ECDSA, one can see that the approach for ECDSA is similar to DHKE, as the given information and searched data can be rearranged to a multiplication solvable by lattice construction. This also is a reason why it is not known whether ECDSA is secure. Note that we do not know whether there is a variant of lattice attack applicable to ECDH.

5.2.2 DHKE with larger Bit-Sizes

Using a key exchange that is secure against the lattice-based attack we mentioned is a viable solution. But there are systems that highly rely on DHKE and do not support chang-

ing the underlying scheme.
Our results showed that the leakage needed to successfully recover a shared secret needs to be relatively large. Over ten leaked most significant bits are needed for bit-sizes above 1024-bit. Commonly used protocols such as TLS are using a bit-size of 2048 and 3072 [KMN19]. With the scaling of the bit-size one needs a high number of leaked bits as well as a higher number of samples fulfilling the leakage. Therefore, DHKE can be considered to be secure against the presented lattice attack if the bit-size is appropriate (greater than 2048-bit) and if the leakage is less than 20-bit and underlying a bit-size of 2048.

5.2.3 DHKE using Ephemeral Keys

With powerful hardware, one can always increase the security parameters or bit-sizes of the used techniques. But there are low-power devices that could be limited to a smaller bit-size. In this case, the private parameter of each party should renew their private and public parameters after a constant number of usages. That property conflicts with our defined attack model for the attack on DHKE, as the model, demands the reuse of one party's value. However, if we limit the number of reuse to more than once, the attacker model continues to function. The problem with a small constant number of reuses is that the leakage is unlikely to result in enough samples, which are usable for the lattice-based attack. When the side-channel attack reveals 10% of usable samples for several key exchanges, one needs to collect at least 100 data points to ensure that there are 10 usable samples. Thus, one could try to set the constant for renewing the values to less than the number of samples needed for a specific leakage of a fixed number of bits. Thereby, an attacker would not be able to collect enough samples in the first place, which would make attacking the system impossible.

5.3 Discussion and Open Problems

In this section, we want to point out some ideas for future work. We further discuss their potential.

5.3.1 Including Errors

Collecting samples can often result in samples containing errors. These errors could be either wrong numbers of leaked zero bits or, in general, faulty leaked bits. Therefore, recent literature [AH21, MBA$^+$21] often included error handling to get an overview of the error resistance of the lattice attacks.

We discovered that one could use the error resistance to some degree by simply taking a sample and creating two new samples, each one with an additional bit to "create" more samples. With this addition, one could simply solve an instance with a given fixed number of samples and enlarge the number of samples by guessing bits for the sample. While this approach does have scaling issues, as it is exponentially expanding the dimension of the matrix, for certain parameter settings, we were able to solve instances that were not solvable with fewer initial samples. At this point, we do not know whether this can be applied to different settings or if it is a coincidence that occurred during our testing. Reducing the minimal number of samples for some settings can be interesting for specific scenarios. Thus, it would be interesting to investigate this behavior in more detail in future work.

5.3.2 Combining Predicates and Guessing

In Section 2.7 we discuss different advanced lattice attacks to which we further added the predicate. One experiment we consider to be interesting was combining guessing bits and apply the predicate. As our results showed, the predicate techniques can solve instances with fewer samples. The previous work of [SETA22] shows that one could guess bits for the nonce which are comparable to our so-called "samples". For this purpose, one could start guessing more bits of our samples while also using a predicate, to reduce the minimum number of leaked bits for certain bit-sizes. As both advancements were applicable to ECDSA as well, this combination technique could yield interesting results.

Bruteforce with Parallelization

While the predicate technique already increases the time needed to solve one instance, the combination with guessing bits will be very costly. Guessing bits on the other hand adds exponential many instances which need to be solved. Therefore, one can parallelize it to reduce the time cost to a minimum.

6 Conclusion

We have seen the potential of adapting recent advancements of lattice attacks to DHKE. We first explained the construction of lattice attacks from a given protocol and how to embed it into a lattice problem, which is then solved by a lattice reduction technique. We first showed in fine granular steps how a lattice attack on ECDSA can be built and which improvements exist. To gain a more precise overview, we discussed advanced lattice reduction techniques, such as sieving and enumeration and recent advanced lattice attacks to provide an overview of the state-of-the-art techniques and how one can construct a lattice attack. With this knowledge, we repeated the same procedure to the lattice attack on DHKE, our main protocol to be investigated.
DHKE in its standard attacker model is not suitable for the lattice attack with a predicate which is why we introduced a new attacker scenario for DHKE. The new scenario enabled us to use the predicate on DHKE with the various techniques out of the framework of [AH21]. We also showed that the lattice attack on DHKE behaves similarly to the attack on ECDSA, as comparable performance gains were observed.
In our evaluation, we showed that the attack advancements on DHKE lead to a similar performance gain as presented in recent works on ECDSA. We then discussed whether this applies to common bit-sizes, as DHKE uses larger bit-sizes than ECDSA. By comparing the performance of different bit-sizes, one can see that the impact of enlarging the bit-size impacts the performance, meaning that attacking DHKE with lattice attacks is harder than attacking ECDSA, as more bits must be leaked. Finally, we discussed how to mitigate the lattice-based attack on DHKE and presented possible advancements for the attack, which could be investigated in future work.

References

[ABF$^+$20] Martin R. Albrecht, Shi Bai, Pierre-Alain Fouque, Paul Kirchner, Damien Stehlé, and Weiqiang Wen. Faster enumeration-based lattice reduction: Root hermite factor k^(1/(2k)) in time k^(k/8 + o(k)). *IACR Cryptol. ePrint Arch.*, page 707, 2020.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.

[AFG$^+$14] Diego F. Aranha, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. GLV/GLS decomposition, power analysis, and attacks on ECDSA signatures with single-bit nonce bias. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 262–281. Springer, 2014.

[AH21] Martin R. Albrecht and Nadia Heninger. On bounded distance decoding with predicate: Breaking the "lattice barrier" for the hidden number problem. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 528–558. Springer, 2021.

[AKLL82] H. W. Lenstra Jr. A. K. Lenstra and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen 261*, (366–389), 1982.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis

Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 601–610. ACM, 2001.

[ANT^{+}20] Diego F. Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom. Ladderleak: Breaking ECDSA with less than one bit of nonce leakage. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 225–242. ACM, 2020.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 10–24. SIAM, 2016.

[BGJ15] Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. *IACR Cryptol. ePrint Arch.*, page 522, 2015.

[BV96] Dan Boneh and Ramarathnam Venkatesan. Hardness of computing the most significant bits of secret keys in diffie-hellman and related schemes. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 129–142. Springer, 1996.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 329–358. Springer, 2020.

[FP85] Ulrich Fincke and Michael E. Pohst. Improved methods for calculating vectors of short length in a lattice. *Mathematics of Computation*, 1985.

[HK17] Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in euclidean norm. In Serge Fehr, editor, *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory* *in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*, volume 10174 of *Lecture Notes in Computer Science*, pages 16–40. Springer, 2017.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 193–206. ACM, 1983.

[Kan87] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

[KMN19] David Cooper (NIST) Kerry McKay (NIST). Guidelines for the selection, configuration, and use of transport layer security (tls) implementations. Technical Report Federal Information Processing Standards Publications (FIPS PUBS) 140-2, Change Notice 2 December 03, 2002, U.S. Department of Commerce, Washington, D.C., 2019.

[Laa16] Thijs Laarhoven. *Search problems in cryptography: from fingerprinting to lattice sieving*. PhD thesis, Mathematics and Computer Science, February 2016. Proefschrift.

[May03] Alexander May. New rsa vulnerabilities using lattice reduction methods, 2003.

[MBA$^+$21] Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk. Raccoon attack: Finding and exploiting most-significant-bit-oracles in TLS-DH(E). In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 213–230. USENIX Association, 2021.

[MH20] Gabrielle De Micheli and Nadia Heninger. Recovering cryptographic keys from partial information, by example. *IACR Cryptol. ePrint Arch.*, page 1506, 2020.

[MSEH20] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger. TPM-FAIL: TPM meets timing and lattice attacks. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2057–2073. USENIX Association, 2020.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1468–1480. SIAM, 2010.

[MW15] Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 276–294. SIAM, 2015.

[NS03] Phong Q. Nguyen and Igor E. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Des. Codes Cryptogr.*, 30(2):201–217, 2003.

[oST13] National Institute of Standards and Technology. Digital signature standard (dss). Technical Report Federal Information Processing Standards Publications (FIPS PUBS) 140-2, Change Notice 2 December 03, 2002, U.S. Department of Commerce, Washington, D.C., 2013.

[PH78] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms overgf(p)and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[Poh81] Michael Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *SIGSAM Bull.*, 15(1):37–44, 1981.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

[SE91] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In Lothar Budach, editor, *Fundamentals of Computation Theory, 8th International Symposium, FCT '91, Gosen, Germany, September 9-13, 1991, Proceedings*, volume 529 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 1991.

[SETA22] Chao Sun, Thomas Espitau, Mehdi Tibouchi, and Masayuki Abe. Guessing bits: Improved lattice attacks on (EC)DSA with nonce leakage. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):391–413, 2022.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.

[The22] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.5)*, 2022. `https://www.sagemath.org`.