Machine-Learning Side-Channel Attacks on the GALACTICS Constant-Time Implementation of BLISS

Soundes Marzougui  
Technical University of Berlin  
Berlin, Germany  
soundes.marzougui@tu-berlin.de

Nils Wisiol  
Technical University of Berlin  
Berlin, Germany  
nils.wisiol@tu-berlin.de

Patrick Gersch  
Technical University of Berlin  
Berlin, Germany  
patrick.gersch@tu-berlin.de

Juliane Krämer  
University of Regensburg  
Regensburg, Germany  
juiliakraemer@ur.de

Jean-Pierre Seifert  
Technical University of Berlin  
Berlin, Germany  
Jean-Pierre.Seifert@external.telekom.de

ABSTRACT
Due to the advancing development of quantum computers, practical attacks on conventional public-key cryptography may become feasible in the next few decades. To address this risk, post-quantum schemes are assumed to be secure against quantum attacks are being developed. Lattice-based algorithms are promising replacements for conventional schemes, with BLISS being one of the earliest post-quantum signature schemes in this family. However, required subroutines such as Gaussian sampling have been demonstrated to be a risk for the security of BLISS, since implementing Gaussian sampling both efficient and secure with respect to physical attacks is challenging.

This paper presents three related power side-channel attacks on GALACTICS, the latest constant-time implementation of BLISS. All attacks are based on power side-channel leakages we identified in the Gaussian sampling and signing algorithm of GALACTICS. To run the attacks, a profiling phase on a device identical to the device under attack is required to train machine learning classifiers. In the attack phase, the leakages of GALACTICS enable the trained classifiers to predict sensitive internal information with high accuracy. We demonstrate the practicality of the attacks by running GALACTICS on a Cortex-M4 and provide proof-of-concept data and implementation for all our attacks.

CCS CONCEPTS
• Security and privacy → Cryptanalysis and other attacks.

KEYWORDS
BLISS, GALACTICS, Gaussian sampler, Machine-Learning, post-quantum cryptography, side-channel analysis

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ARES 2022, August 23–26, 2022, Vienna, Austria  
© 2022 Association for Computing Machinery.  
ACM ISBN 978-1-4503-9670-7/22/08...$15.00  
https://doi.org/10.1145/3538969.3538980

1 INTRODUCTION
BLISS [Ducas et al. 2013], one of the earliest lattice-based signature schemes, attracted significant attention from the scientific community. However, despite the emerging real-world adoption [Steffen et al. 2017] and the efforts targeting efficient and secure implementation [Barthe et al. 2019b], BLISS has been the target of several side-channel attacks [Espitau et al. 2017a; Groot Bruinderink et al. 2016; Pessl et al. 2017a; Tibouchi and Wallet 2020].

Security against side-channel attacks is a major concern for schemes that are meant for real-world deployment. According to Kocher et al. [Kocher et al. 1999], side-channel attacks are considered the main threat to cryptographic algorithms meant for deployment in embedded systems. In such attacks, an attacker does not exploit mathematical weaknesses or invalid behavior of an implementation, but uses the behavior of the device running the implementation to reveal secret data.

In the past, BLISS has been attacked by side-channel attacks based on cache timing of the CPU, where the attacker exploits timing variation caused by the memory management execution to leak sensitive information [Barthe et al. 2019b; Espitau et al. 2017a; Pessl et al. 2017a; Steffen et al. 2017; Tibouchi and Wallet 2020].

Side-channel attacks based on power consumption of embedded devices have also gained much research attention. In this class of attacks, the attacker gains information about sensitive internal data of the device by measuring the power a device consumes during a cryptographic operation with, typically, high sample rate.

Our Contributions. We present three machine-learning-based profiling side-channel attacks against the GALACTICS implementation of BLISS, each allowing a full secret key recovery.

• We identify four leakages in the power analysis of GALACTICS which allow the prediction of internal values with high accuracy. For one case, we demonstrate the superiority of a prediction based on machine learning over linear regression.
Based on the leakages, we demonstrate three attacks.

- In the first attack, we target the constant-time Gaussian sampler [Zhao et al. 2020], and the constant-time sign flip implementation of the signing algorithm. After observing the power consumption of approx. 320 signatures, we are able to fully recover the secret key in a few seconds.
- In the second attack, we restrict the attacker to one of the four leakages. Following the strategy due to Groot Bruinderink et al. [Groot Bruinderink et al. 2016], we demonstrate that – at the expense of more observed signatures – a secret key recovery is still feasible.
- Our third attack demonstrates that even assuming a side-channel resistant Gaussian sampling algorithm, GALACTICS can be attacked, i.e., that a secret key recovery by the method of Tibouchi et al. [Tibouchi and Wallet 2020] is still possible, if the attacker is able to learn information about the sign flipping procedure of the signing algorithm.
- Our three attacks are demonstrated via proof-of-concept experiments which were performed on Cortex-M4 micro controllers.

**Related Work.** The security of most modern lattice-based cryptographic schemes relies on the Learning with Errors problem (LWE). An LWE instance contains the secret vector blinded with a noise vector (error). Usually, the noise vectors are taken from a Gaussian distribution, typically acquiring many samples for a single run of the scheme. The Gaussian sampling process is thus essential to the security of the scheme, as an attacker with knowledge of the samples can compute the solution to the LWE problem and obtain the secret key of the scheme by solving a system of linear equations. Hence, the Gaussian sampling has been considered a potential weakness of lattice-based schemes in general and BLISS in particular.

Sampling from a Gaussian distribution can be based upon a Cumulative Distribution Table (CDT). The CDT sampler includes a table of cumulative distribution function values that covers a finite interval. To output a sample, one generates a uniformly random value and determines the Gaussian sample value by iterating through the CDT until the entry corresponding to the uniform random value is found. A constant-time implementation of a CDT sampler forces the execution of a comparison on all the table’s entries [Bos et al. 2015]. Kim et al. [Kim and Hong 2018] found that high-precision Gaussian samples based on CDT are not only inefficient in terms of required storage space, but also showed their insecurity by demonstrating a single-trace power analysis attack. By recovering the Gaussian samples, they also break the security of the lattice-based scheme employing the sampler.

Ducas et al. [Ducas et al. 2013] proposed a more efficient and secure approach for Gaussian sampling. In a first step, their approach draws from a Gaussian distribution with a small standard deviation. Then, in a second step, these samples are blended with uniformly random values. The deviation of the resulting values from a Gaussian distribution is compensated by an additional rejection condition based on a Bernoulli sample. Due to the blending with uniform values, even if the CDT sample values are obtained by an attack such as the one by Kim et al. [Kim and Hong 2018], the values of the final Gaussian sample cannot be derived. Thus, the secret key cannot be recovered by solving a system of linear equations.

Using cache timing attacks, Groot Bruinderink et al. [Groot Bruinderink et al. 2016] targeted the CDT sampler and Bernoulli rejection (as described in [Ducas et al. 2013]) and demonstrated that the obtained leakage along with public information leads to a full recovery of the secret key. In a similar side-channel attack, Espitau et al. [Espitau et al. 2017b] target the Gaussian sampler and the rejection sampler during the BLISS signing process. Using a branch tracing technique, they reveal the Gaussian samples as well as the Bernoulli samples and demonstrate how to use this information to infer the secret key. The techniques demonstrated in these attacks are based on leakages of Gaussian sampling and Bernoulli rejection (during Gaussian sampling) and do not apply to BLISS-B, an improved variant of BLISS and the default option in strongSWAN [Steffen et al. 2017], an IP- SecV PN library. However, Pessl et al. [Pessl et al. 2017a] presented a new side-channel key-recovery algorithm against both the original BLISS and the BLISS-B variant. Their key recovery attack, while based on the same leakages, also works against BLISS-B and recovers the key using, among other tools, integer programs, maximum likelihood tests, and a lattice-basis reduction. In addition, lattice-based schemes submitted to the National Institute of Standards and Technology (NIST) standardization competition crucially rely on the Gaussian sampler (e.g., FrodoKEM[Alkim et al. 2020] and FALCON [Fouque et al. 2019]). Therefore, the vulnerability of the Gaussian samplers subroutine against side-channel attacks puts these schemes under scrutiny, as revealing the Gaussian samples might reduce the LWE crypto system to a linear system of equations solvable in few seconds.

We conclude that independently of the used techniques for key recovery, the Gaussian sampler and the Bernoulli rejection both pose a risk to the security of BLISS, BLISS-B, as well as other NIST candidates schemes such as FrodoKEM[Alkim et al. 2020] and FALCON [Fouque et al. 2019].

Gaussian and rejection samplers were not the only vulnerabilities of the BLISS scheme. A recent timing attack against BLISS exploits only the bit sign information to achieve full secret key recovery [Tibouchi and Wallet 2020]. In this attack, Tibouchi and Wallet computed parts of the secret key using a maximum likelihood estimation on the space of parameters. To mitigate this attack, they propose to use a constant-time implementation of sign flip, mitigating the leakage based on cache timing.

To mitigate these side-channel attacks, it was suggested to employ a countermeasure consisting of performing a random shuffling (Fisher-Yates random shuffle) [Knuth 1997; Saarinen 2018] after using any non-constant-time sampling scheme. The random shuffle is claimed to mask the relation between the retrieved side-channel information of the samples and the secret [Saarinen 2018]. However, this method cannot completely hide the statistical features of the distributions in the attacked vector. An attacker only requires a marginally larger yet still practical number of samples to rearrange the coordinates and reverse the shuffle operation [Pessl 2016].

Another countermeasure is a constant-time sampling for both Gaussian and Bernoulli distribution via value tables of the cumulative distribution function. An example of this approach is the
FACCT Gaussian sampler due to Zhao et al. [Zhao et al. 2020]. Being an extension of the approach suggested by Ducas et al. [Ducas et al. 2013], the FACCT sampler avoids storing precomputed values for the Bernoulli sampling, using an approach based on polynomial approximation instead.

Taking all known attacks into account, Barthe et al. recently proposed GALACTICS [Barthe et al. 2019b], a constant-time implementation of the BLISS signature scheme. The constant-time implementation of GALACTICS’ Gaussian sampler employs a similar approach as in [Zhao et al. 2020]. The GALACTICS implementation presents not only a constant-time Gaussian sampler but also a constant-time implementation of the whole scheme, including a constant-time implementation of sign flip during signature generation.

Organization. In Sec. 2, we introduce the BLISS signature scheme. Subsequently, in Sec. 3, we present an overview on the attack and attacker model. In Sec. 4, we present our experimental setup and explain the different phases of the attacks. In Sec. 5, we identify attacker model. In Sec. 4, we present our experimental setup and identify the leakages and proposed key recovery attacks.

2 BACKGROUND

2.1 Notation

For any integer $q$, the ring $\mathbb{Z}_q$ is represented by the set $\{-q/2, q/2\} \cap \mathbb{Z}$. Polynomials are defined in the rings $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ or $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ and denoted as bold lower case letters. Vectors are considered column vectors and denoted by bold lower case letters as well, while matrices are denoted by bold upper case letters.

By default, we use the Euclidean $L^2$-norm, i.e., $\|v\| = \sqrt{\sum_i x_i^2}$. By $\lfloor x \rfloor_d$ we denote the $d$ most significant bits of an integer $x$, i.e., $x = \lfloor x \rfloor_d \cdot 2^d + x'$, with $x' \in \{-2^{d-1}, 2^{d-1}\}$.

A lattice $\Lambda$ is a discrete subgroup of $\mathbb{R}^n$. Given $m \leq n$ linearly independent vectors $b_1, \ldots, b_m \in \mathbb{R}^n$, the lattice $\Lambda(b_1, \ldots, b_m)$ is the set of all integer linear combinations of the $b_i$’s, i.e.,

$$\Lambda(b_1, \ldots, b_m) = \left\{ \sum_{i=1}^m x_i b_i \right\},$$

where $b_1, \ldots, b_m$ is the basis of $\Lambda$ and $m$ is the rank. In this paper, we consider full-rank lattices, i.e., with $m = n$. An integer lattice is a lattice for which the basis vectors are in $\mathbb{Z}^n$. Usually, we consider elements modulo $q$, i.e., the basis vectors and coefficients are taken from $\mathbb{Z}_q$.

2.2 Gaussian Sampling

A discrete Gaussian distribution is characterized by two values: the standard deviation $\sigma$ and the mean value $\mu$. A value $x \in \mathbb{Z}$ is sampled with probability

$$\Pr_{X \sim D_\sigma} [X = x] = \frac{\rho(x)}{\sum_{y=-\infty}^{\infty} \rho(y)},$$

where $\rho(y) = \exp\left(\frac{-y^2}{2\sigma^2}\right)$ is the continuous Gaussian function. With $D_\sigma^+$ we denote the non-negative part of $D_\sigma$. There are different generic ways to sample from a discrete Gaussian distribution. An early approach employs the cumulative distribution table (CDT) for sampling [Devroye 1986]. It consists in computing a table $Y$ of cumulative distribution function values of $D_\sigma^+$ that cover a finite interval $[-\tau \sigma, +\tau \sigma]$. The parameter $\tau$ denotes the tail-cut and is chosen such that the probability for drawing from outside the interval is negligible, e.g., less than $2^{-128}$. To output a sample, one generates a uniformly random value from $[-\tau \sigma, +\tau \sigma]$ with 128 bits of precision and returns the index of the first entry in the table greater than the random value. A constant-time implementation of this sampling algorithm requires the execution of a comparison on all the table’s entries [Bos et al. 2015]. Moreover, a 128-bit precision table is expensive in terms of memory storage (e.g., 2730 entries for BLISS-I). A more efficient way, proposed in [Micciancio and Walter 2017], consists of first sampling from a Gaussian distribution $D_\sigma^+$ with a small standard deviation $\sigma_0$. Then, those samples are combined to achieve a larger standard deviation $\sigma$. With this approach, the number of CDT entries can be reduced (e.g., 63 entries for BLISS-I).

An improvement of this approach is used in the constant-time implementation GALACTICS [Barthe et al. 2019b]. Here, the Gaussian sampler outputs a sample $y = Kx + y_u$, where $K$ is a constant, $x$ sampled from $D_{\sigma_0}^+$, and $y_u$ sampled uniformly at random. To obtain samples distributed according to the target discrete distribution $D_\sigma$, a rejection condition is applied. This rejection is denoted as the Bernoulli sampling and rejects the sample $y$ with probability $p = \exp(-y_u/(2Kx))/(2\sigma^2))$. To generate negative samples, one can sample and apply a random sign. This approach decreases the number of required CDT entries further; BLISS-1 needs only 10 entries. Hence, this method is memory-efficient and can be implemented in constant time. The cost is dominated by the rejection sampling step and the generation of the uniform randomness. The calculation of the probability $p$ requires high precision and is thus expensive. GALACTICS computes the polynomial approximation of the exponential rejection probability $p$ with respect to a relative precision (i.e., the precision is calculated based on the Rényi divergence and fixed to 45 bits of relative precision [Barthe et al. 2019b]).

The idea of polynomial approximation was previously introduced in the description of the FACCT sampling algorithm [Zhao et al. 2020]. The authors of FACCT avoid using floating point division in their polynomial approximation because the division operation is known to not be constant time (depending on the employed platform). They use floating point multiplication instead to compute the exponential probability, but this instruction does not guarantee constant-time execution either [Barthe et al. 2019b]. To guarantee constant-time behavior, GALACTICS authors calculate the polynomial approximation using integer polynomials, avoiding floating point multiplication and division altogether.
2.3 The BLISS Signature Scheme

With the current state of the art, BLISS is the most efficient lattice-based signature scheme [Güneysu et al. 2018]. It has been implemented in both software [Ducas et al. 2013] and hardware [Pöppelmann et al. 2014]. BLISS can be seen as a ring-based optimization of the earlier lattice-based scheme of Lyubashevsky [Lyubashevsky 2012], sharing the same "Fiat–Shamir with aborts" structure [Lyubashevsky et al. 2014]. BLISS can be seen as a ring-based optimization of the earlier lattice-based scheme of Lyubashevsky [Lyubashevsky 2012], sharing the same "Fiat–Shamir with aborts" structure [Lyubashevsky et al. 2014]. BLISS can be seen as a ring-based optimization of the earlier lattice-based scheme of Lyubashevsky [Lyubashevsky 2012], sharing the same "Fiat–Shamir with aborts" structure [Lyubashevsky et al. 2014].

Algorithm 1

BLISS signature generation

\[ a_q = \frac{s_2}{s_1} \mod q \]

where the signing polynomials \( s_1, s_2 \in \mathcal{R} = \mathbb{Z}[X]/(X^n + 1) \) are small and sparse. The parameters for BLISS are detailed in the original publication [Ducas et al. 2013].

To sign a message \( m \in \{0, 1\}^* \), we first generate commitment values \( y_1, y_2 \in \mathcal{R} \) with normally distributed coefficients. Then, we compute a hash \( h \) of the message \( m \) together with

\[ u = -a_q y_1 + y_2 \mod q \]

using a cryptographic hash function taking values in the set of elements of \( \mathcal{R} \) with exactly \( \kappa \) coefficients equal to 1 and the others to 0. The signature is the triple \((e, z_1, z_2)\), where

\[ z_1 = y_1 + s_1 c \]

A rejection condition ensures the independence of \( z_1 \) and \( s_1 \) (i.e., \( i \in \{1, 2\} \)). Verification is possible because \( u = -a_q z_1 + z_2 \).

Algorithm 1 BLISS signature generation

\[ a_q = \frac{s_2}{s_1} \mod q \]

where the signing polynomials \( s_1, s_2 \in \mathcal{R} = \mathbb{Z}[X]/(X^n + 1) \) are small and sparse. The parameters for BLISS are detailed in the original publication [Ducas et al. 2013].

To sign a message \( m \in \{0, 1\}^* \), we first generate commitment values \( y_1, y_2 \in \mathcal{R} \) with normally distributed coefficients. Then, we compute a hash \( h \) of the message \( m \) together with

\[ u = -a_q y_1 + y_2 \mod q \]

using a cryptographic hash function taking values in the set of elements of \( \mathcal{R} \) with exactly \( \kappa \) coefficients equal to 1 and the others to 0. The signature is the triple \((e, z_1, z_2)\), where

\[ z_1 = y_1 + s_1 c \]

A rejection condition ensures the independence of \( z_1 \) and \( s_1 \) (i.e., \( i \in \{1, 2\} \)). Verification is possible because \( u = -a_q z_1 + z_2 \).

Algorithm 2 Sampling from \( D_\sigma \)

\[
\begin{align*}
\text{Input} & \quad \text{Target standard deviation } \sigma, \text{ integer } K = \left\lfloor \frac{s}{\sigma_0} + 1 \right\rfloor, \\
\text{Output} & \quad \text{A random integer } y \in \mathbb{Z}^+ \text{ according to } D_K\sigma_0
\end{align*}
\]

where \( \sigma_0 = \frac{1}{\sqrt{2\pi}} \)

1: sample \( x \) from \( D_{\sigma_0}^+ \) using CDT sampler
2: sample \( y_u \in \mathbb{Z} \) uniformly in \([0, ..., K-1]\)
3: sample \( b \) with probability \( \exp(-y_u(y_u+2Kx))/(2\sigma^2)\)
4: if \( b = 0 \) restart [Bernoulli rejection]
5: sample \( a \) uniformly in \([0, 1]\)
6: \( y \leftarrow (-1)^a \cdot (Kx + y_u) \) [sign flip]
7: return \( y \)

2.4 GALACTICS Implementation

An implementation of BLISS with complete timing attack protection – GALACTICS – is presented in [Barthe et al. 2019b]. The GALACTICS implementation relies on integer arithmetic (limited to addition, multiplication, and shifting on 32-bit and 64-bit operands); division instructions and floating point operations were avoided due to their non-constant-time execution that can present serious security challenges [Ducas et al. 2020].

GALACTICS authors avoided the previous implementations of sign flips during the signing and sampling processes, such as the original one by Ducas et al. [Ducas et al. 2013] and the implementation in StrongSwan [Brunner 2008]. These two implementations have conditional branching on the sensitive information about the flipped sign. Therefore, they are not considered constant-time and are vulnerable to timing attacks and branch tracing attacks [Espitau et al. 2017a].

Nevertheless, GALACTICS achieves the same level of efficiency as the original unprotected code [Ducas et al. 2013]. It has been proven experimentally, using the dudect tool by Reparaz et al. [Reparaz et al. 2017], that the implementation is constant-time and secure against microarchitectural side-channel attacks such as [Groot Bruinderink et al. 2016; Pessl et al. 2017b].

3 OVERVIEW OF THE ATTACKS

Through the analysis of four power analysis leakages in GALACTICS on the Cortex M4, we present three profiling side-channel key recovery attacks. We identify three leakages in the Gaussian sampler implementation, namely in the CDT sampler, the Bernoulli rejection, the choice of sign of the sample, and one leakage in the signature generation algorithm affecting the sign flip operation. For the purpose of this study, we consider two identical Cortex M4 CPUs, named Device A and Device B. Device A will be used for profiling, while Device B is the device under attack. In a real-world scenario, an attacker would therefore need access to a clone device which has an architecture identical to the device under attack and can be controlled by the attacker to facilitate the collection of training data for the classifiers, as described below. Moreover, to run the attacks, the attacker must be able to trigger the device under attack to generate signatures using a constant secret key. Our attack is a known-message side-channel attack. However, the attacker does not need to choose the messages to be signed.
Table 1: Comparison of required leaked information (denoted •) for different attacks on GALACTICS [Barthe et al. 2019]. The leaked information on $x$ is the value drawn from the discrete Gaussian distribution $D_μ_σ$ (with tail cut-off), the leaked information on $a$ and $b$ is the value of the uniform choice from $\{0, 1\}$, and the leak on $y_a$ is the information whether $y_a = 0$. For a single signature generation, Alg. 2 is run at least 512 times; Alg. 1 is run at least once.

| Attack | Alg. 2 | Alg. 1 | required signatures |
|--------|--------|--------|---------------------|
| Sec. 6.1 | •      | •      | ~320                |
| Sec. 6.2 | •      | •      | ~2,000              |
| Sec. 6.3 | •      | •      | ~250,000            |

During the profiling phase, we execute the four mentioned functions (CDT sampler, Bernoulli rejection, sign flipping during sampling, and sign flipping during signing) with random input (according to the respective distribution) on our Device A and collect the corresponding power traces in a controlled environment. We label the power traces with the sensitive internal data that we suspect will be leaked (i.e., whether $y_a = 0$ (Alg.2, line 3), the value of $x$ (Alg.2, line 1), the value of $a$ (Alg.2, line 5), and the value of $b$ (Alg. 1, line 4)). With the collected data, we train a total of four MLP classifiers, one for each leakage. By using those artificial neural networks, the profiling phase aims at training the network to learn the leakage of the target device for all possible values of the sensitive variable [Brisfors et al. 2020; Kim et al. 2018; Maghrebi et al. 2016; Sim et al. 2020]. An overview over the leakages and obtained classifiers is given in Tab. 2 in Sec. 5.

In the attack phase, by observing the power traces of the signature generation on Device B, we use the trained classifiers to predict, with high accuracy, the sensitive internal data of the signing algorithm. Together with public information (i.e., $z_1$, $\epsilon$), we recover the secret key in the first attack as the solution to a system of linear equations, and in the second attack as the kernel of a matrix (and, alternatively, using the lattice reduction algorithm LLL). Additionally, in the third attack, we achieve secret key recovery using the maximum likelihood estimation on a set of signatures. We detail on the different key recovery methods in Sec. 6. While the first attack combines all four leakages to achieve key recovery using relatively few power traces, attack two and three only use a single of the four identified leakages. This demonstrates that, at the very least, the leakages on $y_a$ and $b$ need to be addressed to obtain a side-channel resistant implementation.

An overview of the exploited leakage in the attacks is shown in Tab. 1.

4 EXPERIMENTAL SETUP

Workbench. To record the traces for profiling and attacking, we used two STM32F4 microcontrollers, Device A and Device B, mounted on a ChipWhisperer Lite CW308 UFO. As the ChipWhisperer Lite is limited to 24,400 samples per recorded trace, it cannot be used to record the entire power trace of the GALACTICS signature generation algorithm. Instead, we traced each of the four targeted leakages individually by first running GALACTICS on an x86 Ubuntu 20.04 server machine. Then, we use an STM32F4 to rerun sections of the GALACTICS code susceptible to leakage and record power traces. During recording, the ChipWhisperer and the microcontroller both run on the same 7,372,800Hz clock. The sampling rate of the ADC was set to 4 samples/cycle with 10-bit resolution and a 45dB low noise gain filter. Collecting and storing all relevant traces was coordinated using a Python script running on the PC.

Compilation. The GALACTICS source code [Barthe et al. 2019] was provided as a portable C implementation, which makes it suitable for compilation to different architectures. The original benchmarking was done using SUPERCOP [Bernstein and Lange 2008] on an Intel Xeon Platinum 8160-based server (Skylake-SP) with Ubuntu 18.04 and gcc 7.3.0. We compiled GALACTICS using gcc 8.4.0 on an Intel Core i7-6850K CPU (Broadwell E) running Ubuntu 20.04 and the default SUPERCOP [Bernstein and Lange 2008] compiler options (-march=native -mtune=native -03 -fomit-frame-pointer -fwrapv). We adapted the flags and added only those for compiling GALACTICS to ARM for usage on the the target M4 device.

4.1 Adoption of GALACTICS to ARM

When studying the leakage of the CDT sampling algorithm (Alg. 2, line 1), we noticed that GALACTICS uses short-circuit logic in the table look-up, which leads to a run-time dependency on the sampled value. These differences do not reveal information to attackers using a timing side channel, as they cannot observe the timing for a single CDT sample, but only for the entire signing process. However, there is little variance of the entire signing process run-time caused by the CDT sampling, as each signing process invokes the CDT samples at least 1024 times. In contrast, using a power side channel allows for a simple power analysis, since the timing information enables an attacker to reconstruct the chosen CDT sample from the power trace with the naked eye.

To accommodate for this, we replaced all short-circuit operations in the CDT sampler with Boolean logic operators, thereby removing all branching instructions. This does not change the functional behavior, while providing a constant run-time of the CDT sampler function. All attacks in this work are against this hardened implementation of the CDT sampler, and we note that the original implementation’s leakage of the CDT sampler is considerably easier to recognize.

4.2 Profiling Phase

To prepare the profiling, we signed a number of uniformly random messages using random, individual keys. Then, we collected the internal inputs and outputs of the four functions susceptible to leakage of sensitive data (i.e., CDT sampler, Bernoulli rejection, sign flipping during sampling, and sign flipping during signing), including the randomness used, and stored them along with public information about the signing process in the profiling database. With this prepared data, we are able to rerun and analyze the parts of the code susceptible to leakage on the profiling device, Device A.

With the prepared data, we are able to build a list of examples $(x, y) \in \mathbb{R}^t \times Y$, anticipating that the recorded power trace $x$ comprised of $t$ samples reveals information about $y$ from the set...
of sensitive values $Y$ (the exact values in $Y$ depend on the kind of leakage and are shown in Tab. 2). The list of these (noisy) examples $(x, y)$ is split into training, validation, and test set of the multilayer perceptron machine-learning model, with the power traces $x \in \mathbb{R}^t$ acting as the features and the sensitive data $y \in Y$ acting as the labels. Note that the noise is limited to the features, while the labels remain noise-free. To achieve optimal training results, we normalized the feature data to have expectation zero and variance one, as it is a common practice in the training of Multi-layer Perceptron (MLP). The training of the model will result in a function $\hat{f} : X \to Y$ which approximates $f$. We use the accuracy $\Pr_x[f(x) = \hat{f}(x)]$ as metric for the quality of the approximation. To obtain a valid metric, the accuracy is approximated by evaluating $f$ and $\hat{f}$ using the test set, whose examples were not used for training. The choice of model and training parameters can have a decisive influence on the prediction accuracy of the resulting model. We detail on these so-called hyper-parameters in the respective parts of Sec. 5.

4.3 Attack Phase

In the attack phase, we generate a number of uniformly random, attacker-known messages. Then, we collect the trace snippets of all functions susceptible to leakage of sensitive data (i.e., CDT sampler, Bernoulli rejection, sign flip) and store them along with all public information about the signing process. In a real-world attack scenario, the attacker only has access to a complete power trace of the signing process, instead of prepared relevant snippets of it. From a complete trace, however, the position of the relevant snippets can be inferred by a combination of available timing information from GALACTICS (as the implementation is mostly constant time), and correlation of the sampled values with the expectation for the relevant snippet (as the snippets have a distinguished structure).

With the recorded corresponding power trace snippets $x$ from Device B, the attacker is able to use the trained model $\hat{f}$ to obtain a prediction $\hat{y} = \hat{f}(x)$ of the sensitive data with accuracy as given in Tab. 2. In Sec. 6, we explain how to use the leaking information to recover the secret key, in three different attacks.

5 POWER SIDE-CHANNEL LEAKAGE IN GALACTICS

The data required for training is publicly available at https://zenodo.org/record/5101343/files/galactics_attack_data.7z?download=1.

5.1 CDT Sampler Leakage on $x$

The first leakage of the GALACTICS signing procedure studied in this work is the foundation of the Gaussian sampling process: the sampling from the cumulative distribution table (CDT), as invoked in Alg. 2, line 1. The invoked CDT sampler chooses a value $x \in \mathbb{Z}$ according to the distribution $\mathbb{D}_{x, 256}$, that is, having quickly decreasing probability for larger $x$, with $x = 0$ occurring with probability approx. 77%, $x = 1$ with probability approx. 22%, and $x = 3$ with probability approx. 2%. The value is determined by a table look-up based on 128 bits of uniform randomness and then shifted 8 bits to facilitate the multiplication with $K = 256$. In our setup, the tracing of the GALACTICS CDT sampler `sample_bercdt()` including the shift involved 383 cycles, i.e., 1532 measurements using the oscilloscope. The measurements relevant for the recovery of $x$ are shown in Fig. 1.

With the training set built as outlined in Sec. 4.3, an MLP classifier can be trained to predict the CDT samples based on the leakage. We observed that even with no hidden layers in the MLP, i.e., using plain linear regression, the samples can be predicted correctly in 82% of cases, a significant increase over the trivial guessing probability of 77%. By the introduction of a hidden layer, i.e., by also considering nonlinear relationships of power usage and sample value, the accuracy can be increased to 93%. This observation demonstrates that in our case, the usage of MLP is superior to more traditional power analysis methods. Still, even when using a multilayered model, values $x > 3$ are never predicted. Due to the low frequency of these values in the training data, it is difficult to train the model for correct prediction of these values. Nevertheless, even if predicted correctly, the overall accuracy improvement will be negligible.

5.2 Bernoulli Rejection Leakage on $y_u$

After the value $y_u$ is sampled uniformly random from $\{0, \ldots, 255\}$, in Alg. 2, line 3, the Bernoulli rejection condition is applied, which requires the computation of the value $p$, where

$$p = \exp(-y_u(2y_u + 2Kx)/(2\sigma^2)).$$

While we failed to predict the exact value of $y_u$ from the leakage of this computation, our trained classifier can reveal whether $y_u = 0$. We suspect that this is caused by the distinguished Hamming weight of the zero value.

As $y_u = 0$ only occurs in 1/256 = 0.4% of cases, the training of a highly accurate classifier requires some fine-tuning: a classifier always predicting $y_u \neq 0$ already achieves 99.6% accuracy. To prepare the neural network for this task, we adjusted the prediction bias of the output layer to the expected value, 0.4%, and weighted the training example according to their label frequency, i.e., we gave

| Leakage | Alg. | Labels $Y$ | Accuracy on Device B |
|---------|------|------------|----------------------|
| $x$     | Alg. 2 | $\{0, 1, 2, \ldots, 4\}$ | 77% | 82.030% | 93.036% |
| $y_u$   | Alg. 2 | $\{y_u = 0, y_u \neq 0\}$ | 99.6% | 99.105% | 99.953% |
| $a$     | Alg. 2 | $\{a = 0, a \neq 0\}$ | 50.0% | 99.801% | 99.974% |
| $b$     | Alg. 1 | $\{b = 0, b \neq 0\}$ | 50.0% | 99.671% | 100% |

Table 2: Overview over the GALACTICS power side-channel leakage studied in this work and the performance of our predicted models, trained on data from Device A. Trivial prediction accuracy is achieved by always predicting the most likely label, linear accuracy by using a single-layer perceptron, and MLP accuracy by using a multi-layer perceptron. The MLP classifier for $y_u$ was trained using a custom loss function (see Sec. 5.2); the MLP-based classifier for $b$ uses majority vote (see Sec. 5.3).
The third and fourth leakages studied in this work occur when the sign of an integer is flipped. GALACTICS features a constant-time implementation of this operation that mitigates earlier timing and cache side-channel attacks [Tibouchi and Wallet 2020]. Given any integer $x$ and a bit $\lambda \in \{0, 1\}$ indicating if the sign of $x$ will be flipped, the value of $(-1)^\lambda \times x$ is computed using

$$(x \land \lambda) \lor ((-x) \land (1 - \lambda))$$

to avoid any branching instructions. $\lor$ and $\land$ refer to bitwise “OR” and “AND” operations, respectively.

This affects the sign flips $a$ and $b$ respectively in Alg. 2, line 6 and in Alg. 1, line 5. We note that the inputs to the two sign flip operations have different properties. During the Gaussian sampling (Alg. 2, line 6) the input is always positive. Therefore, a flipped sign yields negative results. Moreover, the sign flip operation is called 512 times per signature generation, each using a fresh uniformly random bit indicating the sign. On the other hand, the sign flip operation during the signing process (Alg. 1, line 5) can operate on a positive or negative value for $x$, but is called 512 times using the same sign indicator bit. To account for both situations, we use two separate classifiers.

In both cases, using linear regression, the leakage of the sign flip procedure can be used to predict the sign of the result with accuracy approx. 99.7%, with the prediction of $a$ being slightly more accurate. Similar to the leakage of the CDT sampler in Sec. 5.1, the accuracy can be further improved to about 99.9% by using an additional hidden layer in the classifier.

As the value of $b$ in Alg. 1, line 5, is used 512 times, the prediction accuracy of $b$ can be further improved by conducting a majority vote on the classifications of the 512 individual leakages, resulting in a practically perfect prediction for $b$.

## 6 SECRET KEY RECOVERY

We present three side-channel key-recovery attacks on the recent constant-time implementation of BLISS called GALACTICS. The corresponding source code for all three attacks is publicly available at https://anonymous.4open.science/r/GALACTICS-Side-Channel-Attacks-B6B5.

Each attack uses one or more of the leakages described in Sec. 5 and reveals the complete secret key. We note that to recover the secret key used in the BLISS scheme, it is sufficient to extract $s_1$, the first half of the secret key used in the signing process, as $s_2$ can be recovered through the linear relation:

$$A_s = a_1 \cdot s_1 + a_2 \cdot s_2 \equiv q \mod 2q$$

Hence, throughout this section, we use $s = s_1$ and $z = s_2$. We utilize the indices $i, k$ to refer to the $i$-th coefficient and $k$-th signature, respectively, with $i < n$ and $n = 512$ for BLISS-I.

### 6.1 Attack 1: CDT Samples, Partial Information on Uniform Samples, and Sign Flips

This attack uses all four leakages presented in Sec. 5 and thereby requires relatively few executions of the signing algorithm on the device under attack. After collecting the power traces of approx. 320 runs of the signature generation algorithm, the attacker identifies the relevant snippets. Then, in the first step, an attacker uses the trained classifier of Sec. 5 to predict for each call to the Gaussian sampling algorithm Alg. 2 whether $y_u = 0$, using the power trace snippets of the Bernoulli rejection. As $y_u$ was chosen uniformly at random, we have $\Pr[y_u = 0] = 0.48$. The prediction of $y_u = 0$ was optimized to yield few false positives, cf. Sec. 5.2.
In this attack, we are interested in runs of the Gaussian sampling algorithm (Alg. 2), as in case $y_u = 0$, the signature generated in Alg. 1, line 5, can be written as:

$$
\begin{align*}
  z_{k,i} &= (-1)^{a_{k,i}} \cdot Kx_{k,i} + (-1)^{b_k} \cdot (s, c_k)
\end{align*}
$$

(1)

where $a_{k,i}$ is the corresponding sign flip indicator of the Gaussian samples (Alg. 2, line 6). In a second step, the attacker uses the classifiers of Sec. 5.1 and Sec. 5.3 to obtain high-accuracy predictions for the values $a_{k,i}, b_k, \text{ and } x_{k,i}$ and rearranges Eqn. 1 to:

$$
(s, c_k) = (-1)^{b_k} \cdot (z_{k,i} - (-1)^{a_{k,i}}Kx_{k,i})
$$

(2)

We note that in this representation, the attacker is able to compute a prediction of the right-hand side of Eqn. 2. With the prediction of the right-hand side and the knowledge of the public value $c_k$, the attacker repeats the process to obtain Eqn. 2 for different runs of the signing algorithm. Then, the attacker is able to build a system of linear equations with the coefficients of the secret key $s$ acting as unknowns. To that end, the attacker arranges the values of $c_k$ as rows in a matrix $M$ and the predicted right-hand side values from Eqn. 2 as entries in a vector $r$ to obtain

$$
M \cdot s = r
$$

(3)

A formal algorithmic description of the attack procedure is given in Alg. 3.

**Algorithm 3 GALACTICS Secret Key Recovery Attack**

**Input** Relevant power trace snippets of approx. 320 signatures ($T_{y_u}, T_r, T_c, T_b$) of linearly independent messages executed on the victim’s machine with the same secret key $s$; trained classifiers from Sec. 5; input parameters $\sigma, q, \kappa$ of BLISS-I (as in [Barthe et al. 2019b]).

**Output** Secret key $s$

1. $M \leftarrow [], r \leftarrow [], k \leftarrow 0$
2. for all available traces, indexed by $k$ do
3.  for each $i = 0, n$ do
4.   $\hat{y}_{u,k,i} \leftarrow y_u \cdot \text{classifier}(T_{y_{u,k,i}})$
5.   if $y_{u,k,i} = 0$ then
6.     $\hat{x}_{k,i} \leftarrow x_{k,i} \cdot \text{classifier}(T_{x_{k,i}})$
7.     $\hat{c}_{k,i} \leftarrow c \cdot \text{classifier}(T_{c_{k,i}})$
8.     $\hat{b}_k \leftarrow b \cdot \text{classifier}(T_{b_k})$
9.     add $c_k$ as a row to the matrix $M$
10.    add $(-1)^{\hat{b}_k} \cdot (z_{k,i} - (-1)^{a_{k,i}}Kx_{k,i})$ to $r$
11.    end if
12.   end for
13. end for
14. the secret key $s$ satisfies $M \cdot s = r$ approximately, depending on classifier accuracy.

Even though the classifiers presented in Sec. 5 provide high accuracy, it is unlikely that the system in Eqn. 3 has an exact solution. As the number of $y_{u,k,i} = 0$ in the recorded traces cannot be predicted accurately, the system in Eqn. 3 is likely to be over-determined or under-determined.

To accommodate for this, we use a method that employs the least-squares approach to solve the system. Using the least-squares method, we found that 499 equations are sufficient for full key recovery if no false positive $y_{u,k,i} = 0$ is involved. The reason for the importance of the number of false positives $y_{u,k,i} = 0$ lies in the nature of Eqn. 1, which only applies if $y_{u,k,i} = 0$. The involvement of an equation derived from values when actually $y_{u,k,i} \neq 0$ has thus dramatic influence on the correctness of the approximate solution to the system. The noise introduced by the other classifiers seems less important to the overall stability of the approximate solution.

We also note, as $y_u = 0$ occurs with probability $1/256$ in each of the 512 trials per run of the signing algorithm, with perfect prediction of the $y_u = 0$ condition, 277 runs of the signing algorithm are sufficient to obtain the 499 required equations. Our attack needs significantly more than that, since the classifier for $y_u = 0$ prediction was trained for a low rate of false positives, resulting in a higher rate of false negatives.

### 6.2 Attack 2: Partial Information on Uniform Samples

This attack is an adaptation of the cache side channel attack by Groot Bruinderink et al. [Groot Bruinderink et al. 2016]. The only leakage used is the information whether $y_u = 0$. For different signatures $k$, when $y_{u,k,i} = 0$, the possible values of $y_{k,i}$ (Alg. 1, line 5) are $\{0, \pm K, \pm 2K, \ldots\}$. If additionally, $z_{k,i} \in \{0, \pm K, \pm 2K, \ldots\}$, we can conclude that $(s, c_k) = 0$. In this case, similarly to the previous attack, we construct a matrix $M$ by adding the $c_k$ vector as a new row.

Even though the classifiers presented in Sec. 5 provide high accuracy, it is unlikely that the system in Eqn. 3 has an exact solution. As the number of $y_{u,k,i} = 0$ in the recorded traces cannot be predicted accurately, the system in Eqn. 3 is likely to be over-determined or under-determined. Given the signing of messages operates on a cryptographic hash of the actual message to be signed, we can assume that for different messages, the $c_k$ are uniformly random. Hence, the probability that all collected $c_k$ are linearly independent is very high. In this case, the kernel space of $M$ contains precisely the secret key $s$ if $M$ has full rank 511. We found that when using a noise-free side channel, 1673 signatures are sufficient to recover the key. When fewer signatures are available, this can be compensated by a brute-force search in the (then larger) kernel space of $M$.

To accommodate for potential noisy entries in the matrix $M$ caused by false positive predictions of $y_{u,k,i} = 0$, a larger set of equations can be built. From this larger set, a uniformly random subset of 511 equations can be selected to compute the kernel space. This process can be repeated until 511 noise-free equations were selected, in which case the secret key will be revealed. Using 2000 signatures and the classifier from Sec. 5, with high probability we obtained the full secret key after the first attempts selecting the random row subset of $M$; the total runtime of the attack was below one minute on an x86 Ubuntu 20.04 machine.

**Discussion.** When we expect some noise $\varepsilon$ due to the machine-learning classifier, we can use lattice reduction algorithms such as the LLL algorithm [jr. et al. 1982] to find out the secret vector $s$. Given the attacker built a matrix $M$, we know: $s \cdot M = \nu$. The vector $\nu$ is a short non-zero vector, where $\varepsilon$ of its coefficients are non-zeroes and $n - \varepsilon$ are zeroes. The norm of $\nu$ is highly related to the number of false-positives predicted by our machine-learning
classifier. If our classifier predicts 0 false-positives, then the vector \( \mathbf{v} \) is zero. Otherwise, \( \mathbf{v} \) is non-zero and has a norm at most \( \sqrt{\varepsilon} \cdot n \), with \( \varepsilon \) the number of wrong columns in the matrix \( M \). We reformulate the problem into a \( \gamma \)-approximation version of the shortest vector problem, where one must find a non-zero lattice vector of length at most \( \gamma \cdot \lambda(L) \), where \( \lambda(L) \) denotes the length of the shortest non-zero vector in the lattice \( L \) and \( \gamma \geq 1 \).

The vector \( \mathbf{v} \) is included in the lattice spanned by the rows of \( M \). With the LLL algorithm (in polynomial time), the attacker creates a unimodular matrix \( U \) satisfying: \( U \cdot M = M' \), where \( M' \) is the reduced base of \( M \). With a brute force search, the attacker looks for the secret vector \( s \) in the rows of the unimodular matrix.

Algorithm 4 GALACTICS Secret Key Recovery Attack 2

\[ \begin{align*}
1: & \quad M \leftarrow [ ] \\
2: & \quad \text{for all available traces, indexed by } k \text{ do} \\
3: & \quad \quad \text{for each } i \in \{0, \ldots, n\} \text{ do} \\
4: & \quad \quad \quad y_{u_k,i} \leftarrow \text{classifier}(T_{u_k,i}) \\
5: & \quad \quad \quad \text{if } y_{u_k,i} = 0 \text{ and } z_{k,i} \in \{0, \pm K, \pm 2K, \ldots\} \text{ then} \\
6: & \quad \quad \quad \quad \text{add } c_k \text{ as a row to the matrix } M \\
7: & \quad \quad \text{end if} \\
8: & \quad \text{end for} \\
9: & \quad \text{end for} \\
10: & \quad \text{while } s \text{ not yet found do} \\
11: & \quad \quad M' \leftarrow \text{random selection of 511 rows of } M \\
12: & \quad \quad \text{check if kernel space of } M' \text{ contains the secret key } s \\
13: & \quad \text{end while}
\]  

6.3 Attack 3: Sign Flip during Signature Generation

In this section, we present a third attack that can be mounted even if the sampling process does not leak any data. Following Tibouchi and Wallet [Tibouchi and Wallet 2020], our attack is based solely on the knowledge of \( b_k \) and allows the secret key recovery with approx. 250,000 signatures. In each signature, the sign flip indicator \( b_k \) is sampled once and used 512 times; the classifier outlined in Sec. 5.3 can predict the value of \( b_k \) with high accuracy.

To obtain the secret key, the attack formulates the log-likelihood of secret keys based on the recovered values of \( b_k \) for a large number of signatures. This function admits a unique maximum at the correct secret key \( s_9 \) and can be reliably obtained using gradient ascent.

Using the trained classifier of Sec. 5.3 and the data collected from the device under attack (Device B), we pass the public information of the signing algorithm along with the predicted values of \( b_k \) (Alg. 1, line 5) to an adaptation of the original code to run the attack.

The experimental number of needed signatures is approx. 250,000 signatures for full key recovery. Alternatively, approx. 150,000 signatures are also sufficient to recover 504 coefficients of the secret key. This partial key recovery can be combined with a brute force attack to recover the remaining coefficients.

6.4 Application of the Attacks to the NIST Schemes

FALCON. The authors of FALCON stated in the submissions to the first and second round of the NIST PQC standardization process, that the discrete Gaussian sampling over the integers presents a limitation as it is considered vulnerable against side-channel attacks. However, in the current third round, they claimed that the risk of side-channel attacks is less as their implementation is constant-time. The Gaussian sampler is used in FALCON during the key generation and the signing process. The Gaussian sampler employed in FALCON follows [Zhao et al. 2020]. It relies on a base sampler and Bernoulli rejection subroutines, which we successfully attacked and were able to reveal their outputs with the only knowledge of their power consumption traces. The deviation \( \sigma \) in FALCON, however, is larger than the one in BLISS, which explains the large cumulative distribution table used in the base sampler (18 entries instead of 10 entries). This may affect the accuracy of the classifier when predicting the value of \( x \). In BLISS, the values of \( x \) are between 0 and 4, while in FALCON the values lie in a larger interval. Hence, to train the classifier on \( x \), the attacker needs more traces and more labels. The prediction of \( y_a \) will not be affected but remain as high as 99.95%.

FrodoKEM. The key encapsulation mechanism FrodoKEM has a secret key composed of two parts \( s \) and \( S \) [Alkim et al. 2020]. While \( S \) is the secret key sampled using a cumulative distribution table, \( s \) is a seed chosen randomly during the key encapsulation. Using the leakage described in Sec. 5.1, an attacker will be able to reveal the CDT samples of variables linked to \( S \) linearly. Hence, they can retrieve the secret key \( S \) by solving a system of linear equations. The reason behind this is that the Gaussian sampler used in FrodoKEM relies only on the CDT sampling approach, i.e., the samples are not blended with randomly generated numbers as in [Barthe et al. 2019b; Zhao et al. 2020].

Although the attacker is able to reveal only the part \( S \) of the secret key, they are able - with a single trace and even for ephemeral keys - to calculate the shared secret key using only \( S \). Moreover, we believe that this attack considerably reduces the substantial amount of entropy for a full key recovery. As an example, we consider the variant FrodoKEM-1344 which targets security level 5 of the NIST process. The deviation is 1.4 and the CDT table has 6 entries, which facilitates the training of our machine learning classifier. \( S \) is of size 43,088 bytes, where 21,520 bytes are public and the remaining 21,568 bytes are revealed by our attack. The random secret \( s \) is of size 32 bytes. Hence, 99.92% of the secret key can be revealed by our attack.

7 DISCUSSION AND POSSIBLE COUNTERMEASURES

In this paper, we present three power side-channel key recovery attacks on the latest constant-time implementation of BLISS [Barthe et al. 2019b]. The attacks target four subroutines: the CDT sampling, the Bernoulli rejection, the choice of sign during the Gaussian sampling, and the sign flip during signature generation. We provide detailed leakage analysis based on machine learning techniques. Our analysis results in revealing sensitive data that can be exploited.
to completely recover the secret key. As the targeted subroutines are also used by other lattice-based schemes, e.g., FALCON [Fouque et al. 2019] and FrodoKEM [Alkim et al. 2020], we conjecture they are vulnerable to similar attacks.

The success of the attack in Sec. 6.2 demonstrates that the leakage of the Bernoulli rejection alone suffices to recover the secret key by utilizing approx. 2000 signatures (Tab. 1). Adding the use of leakages on the CDT samples \( x \) and the two sign flip operations, the required number of signatures can be reduced to approx. 320, as demonstrated in the attack of Sec. 6.1. We note that the leakage of \( x \) alone does not jeopardize the security of GALACTICS, as \( x \) is blended with a uniformly random value. However, the attack of Sec. 6.3 demonstrates that the leakage of the sign flip operation outside the Gaussian samples does suffice to recover the secret key.

Therefore, we focus on discussing countermeasures against the attacks on the Bernoulli rejection and sign flip implementations.

A complete side-channel evaluation of the proposed masked implementation [Barthe et al. 2019b] is beyond the scope of this paper. Yet, we believe that the masking techniques described in [Barthe et al. 2019b] can be a powerful countermeasure against our attacks, namely the masked Gaussian sampler function \( \text{GaussGen} \) and the masked random bit generator function \( \text{BitGen} \). We propose partial masking techniques as a countermeasure to the first two attacks (Alg. 3 and 4). For \( y_{u} \), the uniform value added to a random sample from the cumulative distribution table in the process of Gaussian sampling, we can predict if \( y_{u} \) is zero or not. An effective countermeasure would be to sample \( y_{u} \) in two halves, \( y_{u1} \) and \( y_{u2} \), and add them separately to the sample \( x_{k} \). In this case, predicting \( y_{u1} = 0 \) would be useless, as would \( y_{u2} = 0 \). Our attacks can only use leakage information when \( y_{u1} + y_{u2} = 0 \mod 256 \), but could only detect this if \( y_{u1} = y_{u2} = 0 \). This condition is unlikely to be fulfilled, which results in an increasing number of needed signatures, scaling up exponentially with the number of shares. A similar approach can be taken to mask the sign flip as proposed in [Barthe et al. 2019b].

Further research is needed to investigate if and how masking can serve as an effective countermeasure against all exploitable leakage in machine-learning-based side-channel attacks.

ACKNOWLEDGMENTS
The work described in this paper has been supported by the German Federal Ministry of Education and Research (BMBF) (BBDC 2 (ref. 01IS18025A) and FLOQI (ID 16KIS1074)), and the Deutsche Forschungsgemeinschaft (DFG) - SFB 1119 – 236615297. The work described in this paper has been supported by the German Federal Ministry of Education and Research (BMBF) (BBDC 2 (ref. 01IS18025A) and FLOQI (ID 16KIS1074)), and the Deutsche Forschungsgemeinschaft (DFG) - SFB 1119 – 236615297. The work described in this paper has been supported by the German Federal Ministry of Education and Research (BMBF) (BBDC 2 (ref. 01IS18025A) and FLOQI (ID 16KIS1074)), and the Deutsche Forschungsgemeinschaft (DFG) - SFB 1119 – 236615297.

REFERENCES
Erdem Alkim, Joppé W. Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, Ananth Raghunathan, and Douglas Stehila. 2020. FrodoKEM: Learning with errors key encapsulation. https://frodo.org/. Submission to the NIST Post-Quantum Cryptography standardization project, Round 3.

Gilles Barthe, Sonia Belaid, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. 2019a. https://github.com/espitau/GALACTICS.

Gilles Barthe, Sonia Belaid, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. 2019b. GALACTICS: Gaussian Sampling for Lattice-Based Constant- Time Implementation of Cryptographic Signatures, Revisited. In Proceedings of the 2019 ACM SKS\&SAC Conference on Computer and Communications Security (CCS ’19). Association for Computing Machinery, New York, NY, USA, 2147–2164. https://doi.org/10.1145/3319366.3340203

Daniel J. Bernstein and Tanja Lange. 2008. eBACS: ECRYPT Benchmarking of Cryptographic Systems. https://bench.cr.yp.to. [Online; accessed 15-Mar-2022].

Joppé W. Bos, Craig Costello, Michael Naehrig, and Douglas Stehila. 2015. Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem. 553–570 pages. https://doi.org/10.1109/SP.2015.40

Martin Brüss, Sebastian Forssmark, and Elena Dubrova. 2020. How Deep Learning Helps Compromising USM. 135–150 pages. https://doi.org/10.1007/978-3-030-68487-7_9

László Babai. 2008. StrongSwan: The Open Source IPsec-based VPN Solution. https://github.com/strongswan/strongswan. [Online; accessed 15-March-2022].
machine with compact signatures. *J. Cryptogr. Eng.* 8, 1 (2018), 71–84.
Bo-Yeon Sim, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Tae-Ho Lee, Jaeseung Han, Hyojin Yoon, Jihoon Cho, and Dong-Guk Han. 2020. Single-Trace Attacks on Message Encoding in Lattice-Based KEMs. *IEEE Access* 8 (2020), 183175–183191. https://doi.org/10.1109/ACCESS.2020.3029521
Andreas Steffen et al. March 2017. strongSwan: the Open Source IPsec-based VPN Solution (version 5.5.2). https://www.strongswan.org.

Mehdi Tibouchi and Alexandre Wallet. 2020. One Bit Is All It Takes: A Devastating Timing Attack on BLISS’s Non-Constant Time Sign Flips. *Journal of Mathematical Cryptology* 15, 1 (Nov. 2020), 131–142.
R. K. Zhao, R. Steinfield, and A. Sakzad. 2020. FACCT: FAst, Compact, and Constant-Time Discrete Gaussian Sampler over Integers. *IEEE Trans. Comput.* 69, 1 (Jan. 2020), 126–137.