Leak the Secret Key of BBS Short Signature in Relic via Rowhammer

Hi Team,

We are writing to report a potential security issue about the implementation of bbs short signature in the relic library.

Specifically, we cloned the relic repo from github on December 10, 2023 and have analyzed the source code of the Boneh Boyen short signature (or bbs) scheme (more precisely, relic_cp_bbs.c). Based on our analysis, the signature implementation (Line 70 in relic_cp_bbs.c) is vulnerable to fault attacks.

As a proof-of-concept (PoC), we have recovered 190 bits out of 224 secret bits via Rowhammer. As you may know, Rowhammer is software-induced DRAM fault, which can induce bit flips in main memory that runs a commodity system. In our PoC, a malicious user process co-resides in the same system with a victim process that runs the bbs short signature from the relic library. As both processes share the main memory, the adversary can induce bit flips to the secret key before it is used by the victim to sign a message, resulting in a faulty signature. With enough faulty signatures released, the adversary can recover/leak the secret key. For more details of the analysis, PoC and possible countermeasures, they are provided in the following page.

Before our report, RSA implementation in the WolfSSL library and the OpenSSL library have been reported to be vulnerable to Rowhammer-based fault attacks. For the WolfSSL library, its vulnerability is tracked via CVE-2019-19962 [1]. For the OpenSSL library, its RSA vulnerability is tracked by a commit al. [2]. Clearly, we target a different cryptographic algorithm, i.e., bbs signature.

If you have any question or need more details, please let us know. We are looking forward to your reply. Thank you.

Best Regards,

Junkai Liang (Peking University)

Zhi Zhang (The University of Western Australia)

Xin Zhang (Peking University)

Qingni Shen (Peking University)

1 Analysing the BBS Short Signature

In this analysis, we describe the bbs scheme implemented in relic library, based on which, we show how to leak the secret key via Rowhammer.

In the bbs scheme, $\boldsymbol{keygen}(1^{\lambda})$ generates a public key pk and a secret key sk, which correspond to the **cp_bbs_gen** function defined in Line 38 of relic_cp_bbs.c. Particularly, this function builds bilinear groups $\mathbb{G}_1, \mathbb{G}_2$, where $\|\mathbb{G}_1\| = \|\mathbb{G}_2\| = p$ for a constant prime p and their generators g_1, g_2. p is initialized before this function is invoked, i.e., Line 1372 in $relic\_ep\_param.c$. e is defined as a bilinear map : $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Based on p, the function randomly picks $d \leftarrow Z_p^*$ and further computes $q \leftarrow g_2^d, z \leftarrow e(g_1, g_2)$. As such, pk is generated as:

$$pk = (g_1, g_2, e, q, z) \tag{1}$$

Regarding sk, it is defined as:

$$sk = d \tag{2}$$

$\boldsymbol{sign}(d, m)$ is implemented as the **cp_bbs_sig** function from Line 70 of relic_cp_bbs.c. This is a function that takes d and m as inputs, where d is sk and m is an encoded message, and generates a signature σ as follows:

$$\sigma = g_1^{1/(m+d)} \tag{3}$$

$\boldsymbol{verify}(\sigma, m, pk)$ is implemented as a function called **cp_bbs_ver** from in Line 112 of relic_cp_bbs.c that takes a pair of (σ, m) and pk as inputs, and generates 1 (i.e., verification succeeds) if the following equation holds:

$$e(\sigma, q * g_2^m) = z \tag{4}$$

When a single bit flip occurs to d right before the $\boldsymbol{sign}(d, m)$ function is invoked, the generated signature will become as follows:

$$\sigma^{'} = g_1^{1/(m+d^{'})}, \tag{5}$$

where $\sigma^{'}$ is a faulty signature, caused by a faulty secret key $d^{'}$.

Here, we denote $d^{'}$ as $d + \Delta d$ where Δd represents the injected fault. To make Equation (4) hold, Δd must satisfy the following equation:

$$e(\sigma^{'}, q \times g_2^m \times g_2^{\Delta d}) = z \tag{6}$$

When Equation (6) holds, we are able to find out the index of the bit flipped in d and thus recover its original bit. To implement the bit recovery, a **process** function is proposed in Listing 2.2 in Section 2.2.

2 Recovering Secret Key via Rowhammer

Based on the analysis above, we have used Rowhammer to successfully recover 190 bits out of the 224 secret bits in Ubuntu 20.04.6 with Linux kernel 6.1.64, Intel i3-10100 and 8 GB Crucial DDR4-2666. The recovery consists of 2 main steps, i.e., *profiling memory*, and *recovering secret bits*, which are discussed below.

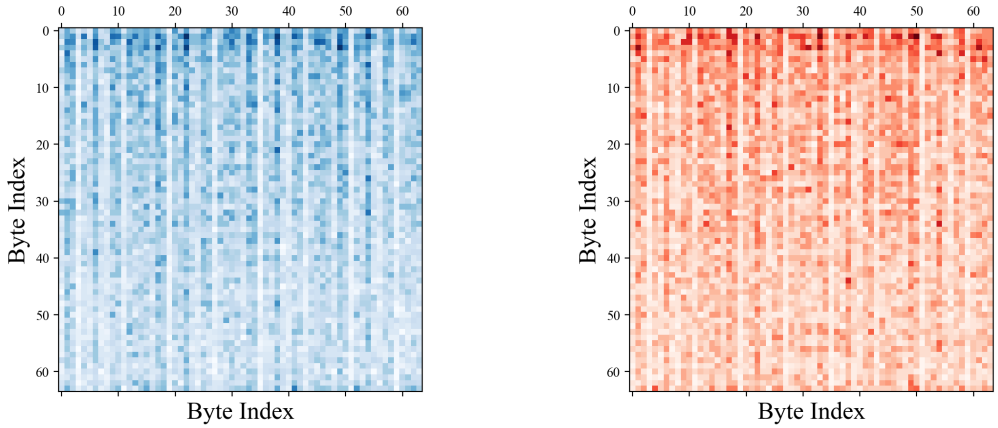

Figure 1: Flippable bit offsets over 4 KB-aligned memory regions via double-sided hammer. Bit flips from 1 to 0 (blue) and bit flips from 0 to 1 (red) can occur at any bit offset.

2.1 Profiling Memory

In this step, we scan the available system memory (via double-sided hammer) to generate a profile of physical pages that are vulnerable to rowhammer bit flips. Based on this profile, we can trigger precise bit flips on the page storing the secret key. Figure 1 shows the distribution of bit-flip offsets accumulated over 4 KB-aligned memory regions. Clearly, bit flips from 1 to 0 (blue) and bit flips from 0 to 1 (red) may occur at any bit offset.

2.2 Recovering Secret Bits

We customized the open-sourced code[1] to lure the victim process into using a vulnerable physical page for storing the secret key. By inducing a single bit flip on the 224 secret-key bits, we can get a faulty signature generated from the victim process. For the signature, we wrote a **process** function below to process it and recover the bit that has been flipped. As only one bit is faulted for the secret key, we can enumerate all possible values for Δd, we use i to indicate the index of a flipped bit and thus compute Δd_i as 2^i or -2^i. In Line 9 of our pseudocode below, we start a loop to enumerate i. In Lines 18 and 25, Δd_i and public parameters z, q are used as input for verification. We use the verification to check if Equation (6) holds. If the verification succeeds, we stop the loop and get index i and its corresponding Δd_i. Given that Rowhammer flips a bit either from 0 to 1 or from 1 to 0, i indicates which bit has been flipped and Δd_i indicates what its original bit value is (i.e., either 0 or 1). After we induce a bit flip for every secret bit, we are able to recover the whole key.

```
1  #include <relic/relic.h>
2  #include <relic/relic_test.h>
3  #include <relic/relic_bench.h>
4
```

[1] https://github.com/a-as-plus-e/FrodoFLIP

```
int process(g1_t s,//signature
            g2_t q,
            gt_t z //q,z are the public parameters
            ){
    for(int i=0;i<bn_bits(sk);i++){ //enumerate all possible indexs of a bit-
        flip fault
        Temporary variables c,g_2,z1,z3;
        //operate the bit difference to pp according to Equation (6)
        bn_set_2b(c,i);
        g2_mul_gen(g_2,c);
        pc_map(z1,s,g_2);
        gt_inv(z1,z1);
        gt_mul(z2,z,z1);
        //If the modified z2(pp) can be verified, the index of a flipped bit
            is targeted
        if(cp_bbs_ver(s, m, sizeof(m), 0, q, z2) == 1){
                bn_set_bit(sk,i,0);
                return 1;
            }
            else{
                gt_inv(z1,z1);
                gt_mul(z3,z,z1);
                if(cp_bbs_ver(s, m, sizeof(m), 0, q, z3) == 1){
                    bn_set_bit(sk,i,1);
                }
                return 1
            }
    }
    return 0;
}
```

In our machine setup, we have generated 1731 faulty signatures, based on which, 190 unique bits out of the 224 secret bits have been recovered and can be further used to infer remaining bits of the secret key sk. Figure 2 shows the bits that have been recovered and more bits are expected to be recovered.

3 Mitigating the Fault-Injection Attack

An effective mitigation is to *verify the signature after signing a message*. This countermeasure has been used by the OpenSSL and the WolfSSL library to mitigate the Rowhammer attack against the RSA implementation [2, 3]. To this end, when updating the bbs scheme implementation, verification is added right after the signature generation. More precisely, we can use the aforementioned **cp_bbs_ver** function. If the verification fails, the **cp_bbs_sign** will not generate a signature. By doing so, the fault can be detected and no faulty signature can be leveraged by the adversary. Compared to signing, verification is much faster, thus making the performance overhead acceptable.

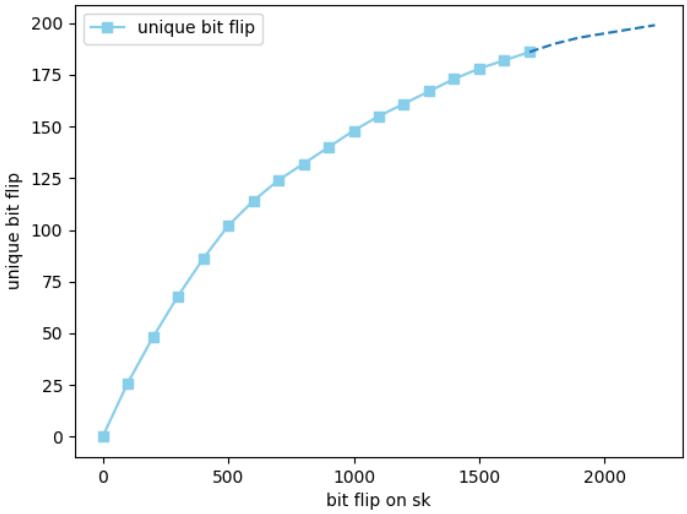

Figure 2: The number of bit flips occured to the secret key sk. The dotted and solid line denotes the observed number bit flips on sk. The dashed line represents the expected bit flips when a bit flip is assumed uniformly random.

References

[1] *CVE-2019-19962*. Available from MITRE, CVE-ID CVE-2019 19962. 2019.

[2] *Openssl commit: Add a protection against fault attack on message v2*. 2018. URL: https://github.com/openssl/openssl/pull/7225/commits/02534c1ee3e84a1d6c59a887a67bd5ee81bcf6cf.

[3] *WolfSSL Release 5.5.0 (Aug 30, 2022)*. 2022. URL: https://github.com/wolfSSL/wolfssl/releases/tag/v5.5.0-stable.