Experiment Evaluation of Fault Injection Attack of BBS Signature

Based on the analysis, 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.

0.1 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.

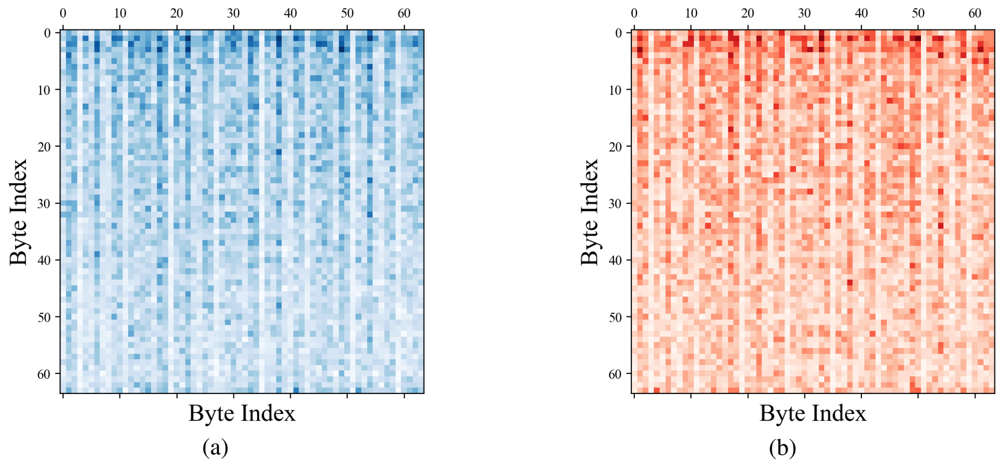

Figure 1: Flippable bit offsets over 4 KB-aligned memory regions via double-sided hammer. Bit flips from 1 to 0 (blue) and bit flips from 0 to 1 (red) can occur at any bit offset.

0.2 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

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

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
5  int process(g1_t s,//signature
6              g2_t q,
7              gt_t z //q,z are the public parameters
8              ){
9      for(int i=0;i<bn_bits(sk);i++){ //enumerate all possible indexs of a bit-
           flip fault
10         Temporary variables c,g_2,z1,z3;
11         //operate the bit difference to pp according to Equation (6)
12         bn_set_2b(c,i);
13         g2_mul_gen(g_2,c);
14         pc_map(z1,s,g_2);
15         gt_inv(z1,z1);
16         gt_mul(z2,z,z1);
17         //If the modified z2(pp) can be verified, the index of a flipped bit
               is targeted
18         if(cp_bbs_ver(s, m, sizeof(m), 0, q, z2) == 1){
19                 bn_set_bit(sk,i,0);
20                 return 1;
21             }
22             else{
23                 gt_inv(z1,z1);
24                 gt_mul(z3,z,z1);
25                 if(cp_bbs_ver(s, m, sizeof(m), 0, q, z3) == 1){
26                     bn_set_bit(sk,i,1);
27                 }
28                 return 1
29             }
30     }
31     return 0;
32 }
```

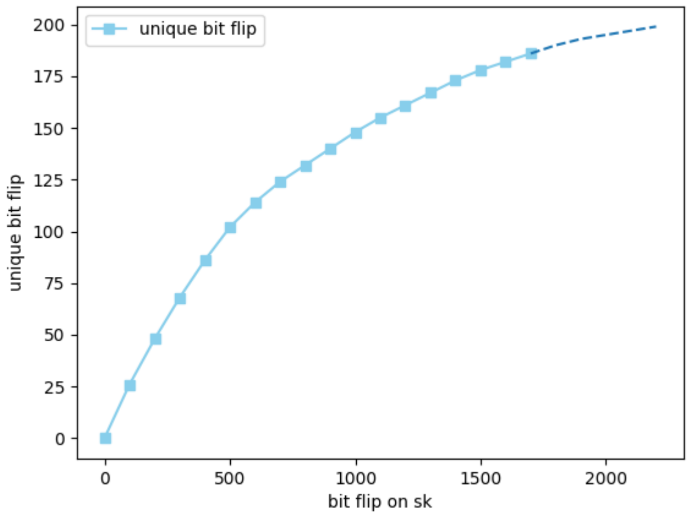

Figure 2: The number of bit flips occured to the secret key sk. The dotted and solid line denotes the observed number bit flips on sk. The dashed line represents the expected bit flips when a bit flip is assumed uniformly random.

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.

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.