Theoretical Analysis of BBS 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 σ' 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 function related to Equation 6 is included in experiment.pdf in this repo.