



$\text{Kg}(1^k)$	$\text{Sign}(sk, m)$	$\text{Vfy}(pk, m, \sigma)$
1 : $sk \leftarrow \mathbb{Z}_p$	1 : $r \xleftarrow{\$} \mathbb{Z}_p; I \leftarrow g^r$	1 : $R \leftarrow g^s \cdot pk^{-e}$
2 : $pk \leftarrow g^{sk}$	2 : $e \leftarrow \text{H}(pk    I    m)$	2 : <b>if</b> $\text{H}(pk    R    m) = e$
3 : <b>return</b> $(pk, sk)$	3 : $s \leftarrow r + sk \cdot e \bmod p$	3 : <b>return</b> 1
	4 : <b>return</b> $\sigma = (s, e)$	4 : <b>else return</b> 0

**Answer 2: Yes, key-prefixed short Sentences are!**



# Summary of Our Results

# Research Questions







Are short snippets secure against preprocessing attacks?

**Caveats:**



- Not a **standardized** implementation
- Preprocessing attacker is **time-bounded** (large enough for practical attacks)
- Complex proof technique: **compression argument**

recorded in  $\mathcal{L}$  before a query of the form  $H(f)(x)$  is ever issued. If  $f$  is new then we call  $\text{Sign}(f)$  before querying the random oracle. Now define a value  $\hat{L} \in \mathcal{L}$  as the set of tuples  $(\hat{L}, \hat{L}, \hat{L}) \in \mathbb{G} \times \mathbb{G}_2^2 \times \mathbb{G}_2$  such that  $\hat{L}$  has exactly one nonzero element. Now we call a random oracle query  $x = (f)(m)$  “bad” if  $H(x) = -\hat{L}$  where the tuple  $(f, \hat{L}, \hat{L}) \in \hat{\mathcal{L}}$  has already been recorded and the nonzero element of  $\hat{L}$  is  $\hat{L}$  (Recall that if there were two recorded tuples  $(f, \hat{L}, \hat{L})$  and  $(f, \hat{L}, \hat{L})$  then our algorithm would have already found a BRIDGE<sup>+</sup> instance). Thus, the probability each individual query is “bad” is at most  $\frac{1}{p}$  and we can use union bounds to upper bound the probability of any “bad” query as  $\text{Pr}[\text{BadQuery}] \leq \frac{1}{p}$ .  $\square$

### C.2 Missing Proof from Section 5

**Reminder of Theorem 7.** Let  $p > 2^{28}$  be a prime number and  $N \in \mathbb{N}$  be a parameter. Let  $(A_{\text{gen}}, A_{\text{ch}})$  be a pair of generic algorithms for  $\mathbb{G}$  with an encoding map  $\tau : \mathbb{G} \rightarrow \mathbb{G}$  such that  $A_{\text{ch}}$  outputs an  $S$ -bit list and  $A_{\text{gen}}$  takes as input  $q^* \in \mathbb{G}_2^N$  queries to the generic group oracle.

$$\text{Pr}[\text{BridgeChal}_{A_{\text{gen}}, A_{\text{ch}}}^{\text{Schnorr}}(N) = 1] \leq \frac{1}{p} \left( \frac{N \log N}{\log p} + \frac{N^2}{\log p} + \frac{N^2}{\log p} \right).$$

where the randomness is taken over the selection of  $\tau$ , the random coins of  $A_{\text{gen}}$  and the random coins used by the challenger in the bridge game (the list  $m_i = A_{\text{ch}}^{\text{gen}}(g)$  is selected independently of the random coins used by the challenger). In particular, if  $q^* \geq 10N(1 + 2 \log p)$  and  $S \geq 10 \log(N)$ , then

$$\text{Pr}[\text{BridgeChal}_{A_{\text{gen}}, A_{\text{ch}}}^{\text{Schnorr}}(N) = 1] \leq \frac{1}{p} \frac{10N \log(N) + N^2}{p}.$$

**Proof of Theorem 7.** We follow the same idea from [33, 48, Theorem 2], which shows the relationship between the size of the list  $m$  and the probability that one can win the multi-user bridge-finding game. For that a single  $p$  is good if  $(A_{\text{gen}}, A_{\text{ch}})$  wins the multi-user bridge-finding game with probability at least  $c/2$  on  $\tau$ . Then, it is  $p$  good if we have

$$\text{Pr}[\text{BridgeChal}_{A_{\text{gen}}, A_{\text{ch}}}^{\text{Schnorr}}(N) = 1] \geq \frac{c}{2}.$$

where the probability is taken over the selection of  $\tau = (\tau_1, \dots, \tau_N) \in \mathbb{G}_2^N$ , the random coins of  $A_{\text{gen}}$  and the random coins used by the challenger in the bridge game (the list  $m_i = A_{\text{ch}}^{\text{gen}}(g)$  is selected independently of the random coins used by the challenger).

Let  $T = (\tau_1, \tau_2, \dots)$  be the set of good labeling maps. Then by a simple double averaging argument ([48], Lemma A.10), we can show that the probability of finding a bridge-finding game with  $A_{\text{gen}}$  and  $A_{\text{ch}}$  is at least  $\frac{c}{2}$  if we can win the multi-user bridge-finding game with probability at least  $c/2$  on  $\tau$ . Then, it is  $p$  good if we have

$$\text{Pr}[\text{BridgeChal}_{A_{\text{gen}}, A_{\text{ch}}}^{\text{Schnorr}}(N) = 1] \geq \frac{c}{2}.$$

#### Handling $\text{Re}(h, h_1)$

- (1) If either  $h_1$  or  $h_2$  is not in the image of  $\tau$ , reply  $\perp$  and continue to the next query.
- (2) If either of  $h_1$  or  $h_2$  is not in the table, then this is a fresh query input. For each such argument  $s \in \{h_1, h_2\}$ , add the pair  $(\text{Sign}(s), s)$  to the table and write  $s$  to the encoding using  $\log(p - f)$  bits, where  $f$  is the number of labels already in the table.
- (3) Otherwise, look up the tuple  $(f, h_1, h_2)$  in the table where  $f_i = f_i(\tau_1, \dots, \tau_N)$  and  $f_j = f_j(\tau_1, \dots, \tau_N)$  are linear polynomials of  $N$  indeterminates  $X_1, \dots, X_N$ , and compute  $f_1 + f_2$ .
- (4) If  $(f_1 + f_2, \text{Re}(h_1, h_2))$  is already in the table, simply reply with  $\text{Re}(h_1, h_2)$ .
- (5) If  $(f_1 + f_2, \text{Re}(h_1, h_2))$  is not in the table, then add  $\text{Re}(h_1, h_2)$  to the encoding and reply with  $\text{Re}(h_1, h_2)$ . Write  $\text{Re}(h_1, h_2)$  into the encoding using  $\log(p - f)$  bits, where  $f$  is the number of labels already in the table.
- (6) If  $\text{Re}(h_1, h_2)$  is in the table but its corresponding discrete log value is in the table in a linear polynomial  $f = f(X_1, \dots, X_N)$  such that  $f$  is not identical to  $f_1 + f_2$  (i.e., the coefficients for  $X_i$ 's are not all the same), then encode the reply to this query as a pointer to the table entry  $(f, \text{Re}(h_1, h_2))$  and add this pointer to the encoding. Then use the equation  $f = f_1 + f_2$  to derive an equation  $X_i = g(X_1, \dots, X_N)$  for some  $j$  and a linear polynomial  $g$  of  $X_1, \dots, X_N$  except for  $X_j$  and replace  $X_j$  by  $g(X_1, \dots, X_N)$  in the table.

In this case, we successfully found a BRIDGE<sup>+</sup> instance, so stop the execution of the algorithm and indicate the early stop by writing the actual number of queries  $q^* \leq q_{\text{hit}}$  into its placeholder above.

#### Handling $\text{In}(h)$

- (1) If  $h$  is not in the image of  $\tau$ , reply  $\perp$  and continue to the next query.
- (2) If  $h$  is not in the table, then this is a fresh query input. Add the pair  $(\text{Sign}(h), h)$  to the table.
- (3) Otherwise, look up the tuple  $(f, h)$  in the table where  $f = f(X_1, \dots, X_N)$  is a linear polynomial of  $N$  indeterminates  $X_1, \dots, X_N$ , and compute  $f$ .
- (4) If  $(f, \text{In}(h))$  is already in the table, simply reply with  $\text{In}(h)$ .
- (5) If  $(f, \text{In}(h))$  is not in the table, then add  $\text{In}(h)$  to the encoding and reply with  $\text{In}(h)$ . Write  $\text{In}(h)$  into the encoding using  $\log(p - f)$  bits, where  $f$  is the number of labels already in the table.
- (6) If  $\text{In}(h)$  is in the table but its corresponding discrete log value is in the table in a linear polynomial  $f = f(X_1, \dots, X_N)$  such that  $f$  is not identical to  $f$ , then add  $\text{In}(h)$  to the encoding and reply with  $\text{In}(h)$ . Write  $\text{In}(h)$  into the encoding using  $\log(p - f)$  bits, where  $f$  is the number of labels already in the table.

where  $q^*$  denotes the number of queries to the signing oracle and the randomness is taken over the selection of  $\tau$  and the random coins of  $A_{\text{ch}}$  (the list  $m_i = A_{\text{ch}}^{\text{gen}}(g)$  is selected independently of the random coins used by the challenger). In particular, if  $q^* \geq 10N(1 + 2 \log p)$  and  $S \geq 10 \log(N)$ , then

$$\text{Pr}[\text{BridgeChal}_{A_{\text{gen}}, A_{\text{ch}}}^{\text{Schnorr}}(N) = 1] \leq \frac{1}{p} \left( \frac{N \log N}{\log p} + \frac{N^2}{\log p} + \frac{N^2}{\log p} \right).$$

**Proof of Theorem 6.** Given a generic adversary with preprocessing  $(A_{\text{gen}}, A_{\text{ch}})$  attacking bridge-finding games, we construct the following efficient generic algorithm with preprocessing  $(A_{\text{gen}}, A_{\text{ch}})$  which takes as input the  $(1-n)$ -user BRIDGE<sup>+</sup> finding game  $\text{BridgeChal}_{A_{\text{gen}}, A_{\text{ch}}}^{\text{Schnorr}}(N, \delta)$ .

**Algorithm  $(A_{\text{gen}}, A_{\text{ch}})$ .** The algorithm is given  $p = (1, \delta, \delta, \delta) \in \mathbb{G}_2^4$  as input.

1.  $A_{\text{gen}}$  simply runs  $A_{\text{ch}}^{\text{gen}}$  to generate an  $S$ -bit list  $m$ .
2.  $A_{\text{ch}}$  initializes the set  $R_{\text{gen}} = \emptyset$  which stores the random coins in past queries (not observed during online processing).
3.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
4.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
5.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
6.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
7.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
8.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
9.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
10.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
11.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
12.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
13.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
14.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
15.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
16.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
17.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
18.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
19.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
20.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
21.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
22.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
23.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
24.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
25.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
26.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
27.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
28.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
29.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
30.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
31.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
32.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
33.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
34.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
35.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
36.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
37.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
38.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
39.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
40.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
41.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
42.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
43.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
44.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
45.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
46.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
47.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
48.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
49.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
50.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
51.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
52.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
53.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
54.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
55.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
56.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
57.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
58.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
59.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
60.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
61.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
62.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
63.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
64.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
65.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
66.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
67.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
68.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
69.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
70.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
71.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
72.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
73.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
74.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
75.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
76.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
77.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
78.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
79.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
80.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
81.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
82.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
83.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
84.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
85.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
86.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
87.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
88.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
89.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
90.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
91.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
92.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
93.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
94.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
95.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
96.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
97.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
98.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
99.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .
100.  $A_{\text{ch}}$  takes the list  $m$  and initializes the list  $\mathcal{L} = \{(1, \delta, \delta, \delta), (\delta, \delta, \delta, \delta)\}$  for  $i = 1, \dots, N$ .





# Summary of Our Results

## Research Questions



Are **short** Schnorr signatures secure against **preprocessing attacks**?

- ▷ **Answer 3:** Yes, “short” version of **standardized implementations** of Schnorr signatures are secure!