On methods of shortening ElGamal-type signatures

Liliya Akhmetzyanova, Evgeny Alekseev, Alexandra Babueva
and Stanislav Smyshlyaev

CryptoPro LLC, Russia
{lah, alekseev, babueva, svs}@cryptopro.ru

Abstract

Development of signature schemes providing short signatures is a quite relevant non-trivial challenge for cryptographers. Since the late 1980's many short signature schemes have been proposed. The most perspective schemes are multivariate schemes and schemes based on Weil pairing. Unfortunately, the cryptographic tools used in these schemes are still not supported by most cryptographic software that complicates their effortless use in practice.

In the current paper we investigate the opportunity of shortening the standard ElGamal-type signatures. We propose three methods of shortening signatures (for any ElGamal-type schemes such as ECDSA, GOST and SM2) and analyze how applying these methods affects the security. Applying all three methods to the GOST signature scheme with elliptic curve subgroup order q, $2^{255} < q < 2^{256}$, can reduce the signature size from 512 to 320 bits. The modified scheme provides sufficient security and acceptable (for non-interactive protocols) signing and verifying time.

Keywords: short signature scheme, ElGamal-type signature scheme, GOST, provable security.

1 Introduction

A signature scheme is one of the most widely used cryptographic protocol in practice. It is a self-supporting protocol replacing a handwritten signature and is used as a primitive in a huge amount of multiple protocols (e.g. TLS Handshake [1] and IKEv2 [2]). Therefore, the operational characteristics of signature scheme such as sizes of keys and signature, time complexity of signing and verifying, are crucial for applications. Although all parameters are important, in the current paper we focus only on the size of signature values.

One of the applications requiring short signatures is the systems where a human is asked to manually key in the signature. For example, product registration systems often ask users to key in a signature provided on a CD label. Also, the size of signature influences the requirements on a channel capacity which may be essential for low-bandwidth communication environments (e.g. Internet of Things and QR codes).

1.1 Related works

Due to the relevance of the considered issues many short signature schemes have been proposed since the late 1980's. One of these schemes is known as a BLS signature scheme [3] based on Weil pairing and providing 160 bits signatures with sufficient for practice security.

Other type of short signature schemes is multivariate schemes based on Hidden Field Equations (HFE) such as Quartz [5], Gui [10], SFLASH [4], UOV [12], Rainbow [13]. Although several of these schemes have been broken due to newly developed attacks (see [6, 7]), a number of multivariate schemes such as UOV, Rainbow withstood cryptanalysis (for suitable parameter sets) for more than 20 years. Also in 2016 the work [11] proposed technique reducing the signature size of almost every multivariate signature scheme by 10 to 15 % without increasing the key sizes or slowing down the scheme significantly. The authors claim that by applying their technique to the Gui signature scheme they obtain signatures of size only 110 bits, «which are the shortest signatures of all existing digital signature schemes». However, the scheme has relatively large public key (about 100 KByte) and slow verification time for the smallest signatures.

In light of the above, the BLS signature scheme is treated as a most favourable solution. Unfortunately, currently Weil pairing is still a non-typical cryptographic tool requiring generation and consequent deep analysis of so called «pairing-friendly» curves. Hence, not any cryptographic software supports this type of curves (unlike typical well-investigated elliptic curves used in standard signature schemes such as ECDSA and GOST). Therefore, the task to provide shorter signatures using typical cryptographic primitives is relevant.

1.2 Our contribution

In the current paper we consider the standard ElGamal-type signature schemes [14] (particularly GOST [19, 20, 21, 22]) with two-component signatures $\overline{r}\|\overline{s}$, where $\overline{r}$ and $\overline{s}$ are $\lceil \log_2 q \rceil$-bit strings (here q is the prime order of the used elliptic curve subgroup), and propose three methods of shortening this type of signature:

- The first method is to replace an internal function f (a mapping from a random elliptic curve point to an integer $r \in \mathbb{Z}_q$) by the hash function with truncated output that implies the reduction of the $\overline{r}$ component size.

– The main idea behind the second method is to make the certain bits of the $\overline{r}$ signature component to be constant and then to cut out them. This method leads to increasing signing time.

– The last method is to directly truncate signature ($\overline{r}$ and/or $\overline{s}$ component). This methods leads to increasing verification time.

All these methods are independent and can be applied together in any combination.

The idea to use hash function for the ElGamal-type signature shortening is also briefly mentioned in [29]. Unlike our first method, the method presented in [29] modifies the original signature scheme significantly due to hashing the message and the $\overline{r}$-component together. Moreover, from what we understand, the authors propose to use the same hash function as for message processing in the original signature scheme. Therefore, it is not clear by what exact means signature shortening is made since the hash function output is usually as long as the original components. Even if truncation of the hash output is implicitly supposed to be done, authors present only asymptotic security results (in terms of polynomial adversaries and negligible success probabilities). However, such a result cannot be used for choosing hash output length securely for practical application and concrete security bounds should be presented. The paper [11] proposes similar technique as in the last method but for multivariate signature schemes specifically. Unlike our method, signature verification in [11] does not imply signature recovering.

We analyse how applying the methods affects the security by obtaining concrete SUF-CMA-security bounds for modified schemes in the random oracle model. The first method changes the internal structure of the base scheme (function f) and in fact provides a new instance of the ElGamal-type signature scheme. For presentation purposes we obtain security bounds for the modified GOST signature scheme (named GOST-H) only, although we believe that the proof can be easily generalized. Using standard techniques we show that the hardness of elliptic curve discrete logarithm problem (ECDLP) and standard security properties of the hash function implies SUF-CMA-security of the GOST-H signature scheme. The second and the third methods are considered in general: for them we reduce the security of the base unmodified scheme to the security of the corresponding modified signature scheme. Applying all three methods to the GOST signature scheme with EC subgroup order q, $2^{255} < q < 2^{256}$, can reduce the signature size from 512 to 320 bits. The modified scheme provides sufficient security and acceptable for non-interactive protocols signing (≈ 6 seconds) and verifying

(≈ 3 seconds) time.

1.3 Paper organization

The remainder of the paper is organized as follows. In Section 2 basic definitions and notations are introduced. Section 3 introduces ElGamal-type signature schemes and describes the main object of the research — three methods of shortening signatures of such type. In Section 4 we formally define basic security notions for signature schemes and accompanying primitives. Section 5 is devoted to the security analysis of the proposed methods. We draw our conclusions in Section 6. Detailed proofs of our theorems are relegated to the appendices due to space limitations.

2 Basic notations and definitions

By $\{0,1\}^s$ we denote the set of s-component bit strings and by $\{0,1\}^*$ we denote the set of all bit strings of finite length including the empty string. For bit strings a and b we denote by $a\|b$ their concatenation. Let $|a|$ be the bit length of the string a.

For a bit string u and a positive integer $l \leqslant |u|$ let $\mathsf{msb}_l(u)$ ($\mathsf{lsb}_l(u)$) be the string consisting of the l rightmost (leftmost) bits of u. For integer $r \geqslant 0$ let $\mathsf{str}(r)$ (or just $\overline{r}$) be the ($\lfloor \log_2(r) \rfloor + 1$)-bit representation of $r > 0$ with the least significant bit on the left and zero bit if $r = 0$. For a bit string u let $\mathsf{int}(u)$ be the integer r such that $\mathsf{str}(r) = u$.

If p is a prime number then the set $\mathbb{Z}_p$ is a finite field with characteristic p. We assume the canonic representation of the elements in $\mathbb{Z}_p$ as a natural number in the interval $[0 \ldots p-1]$. Each non-zero element x in $\mathbb{Z}_p$ has an inverse $1/x$. We define $\mathbb{Z}_p^*$ as the set $\mathbb{Z}_p$ without zero element.

We denote the group of points of elliptic curve over the field $\mathbb{Z}_p$ as $\mathbb{G}$, the order of the prime subgroup of $\mathbb{G}$ as q and elliptic curve point of order q as P. We denote the group generated by P as $\langle P \rangle$ and neutral element in $\mathbb{G}$ as O.

For any set A and B let $Func(A, B)$ be the set of all mappings from A to B. If the value s is chosen from a set S uniformly at random, then we denote $s \overset{\mathcal{U}}{\leftarrow} S$.

If the variable x gets the value val then we denote $x \leftarrow val$. Similarly, if the variable x gets the value of the variable y then we denote $x \leftarrow y$. If the variable x gets the result of a probabilistic algorithm A we denote $A \xrightarrow{\$} x$ ($x \xleftarrow{\$} A$). If we need to emphasize that A is deterministic than we denote it by $A \to x$ ($x \leftarrow A$). The event when A returned value val as a result is denoted by $A \to val$.

We define security properties using the notion of «experiment» played between a challenger and an adversary. The adversary and challenger are modelled using consistent interactive probabilistic algorithms. The challenger simulates the functioning of the analysed cryptographic scheme for the adversary and may provide him access to one or more oracles. The parameters of an adversary $\mathcal{A}$ are its computational resources (for a fixed model of computation and a method of encoding) and oracles query complexity. The query complexity usually includes the number of queries. Denote by $\mathsf{Adv}_{S}^{\mathrm{M}}(\mathcal{A})$ the measure of the success of the adversary $\mathcal{A}$ in realizing a certain threat, defined by the security notion M for the cryptographic scheme S. The formal definition of this measure will be given in each specific case.

3 Three methods of shortening

A signature scheme consists of three algorithms $\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf}$ such that: algorithm KGen generates secret signing key sk and public signature verification key pk; algorithm Sig takes as input a signing key sk and message m and generates a signature sgn for message m; deterministic algorithm Vf takes as input verification key pk, message m and candidate signature sgn and outputs 1 (accept) or 0 (reject). It is required that for every $(\mathsf{pk}, \mathsf{sk})$ outputted by KGen and every message m it holds that

$$\mathsf{Vf}(\mathsf{pk}, m, \mathsf{Sig}(\mathsf{sk}, m)) = 1.$$

The GenElGamal framework is introduced in [14].

We follow the notations of [19] and change the definition in [14] in the following way: we represent the signature as the concatenation of vectors $\overline{t}$ and $\overline{s}$ instead of pair of elements in $\mathbb{Z}_q$, we denote r as k, h as e, t as r, x as d, X as Q. Moreover, we denote the GenElGamal signature scheme as GenEG.

The signature generated by the GenEG scheme is represented as $\overline{r}\|\overline{s}$, where $r, s \in \mathbb{Z}_q$, thus, its size is at most $2\lceil \log q \rceil$. The following sections introduce three methods of shortening signature size, we call the schemes obtained by applying these methods $\mathsf{GenEG\text{-}H}$, $\mathsf{GenEG_S}$ and $\mathsf{GenEG^V}$ respectively.

3.1 $\mathsf{GenEG\text{-}H}$ scheme

The r component in all GenEG schemes is computed as the result of applying function f to point R. The idea behind the first method of shortening signature is to modify function f by splitting it into two functions. At first we apply the compression function H_2 to the x-coordinate of point R and then we represent the result as an element in $\mathbb{Z}_q^*$. Note that the bit representation of $\overline{r}$ will be shorter due to compression.

The modified function f is defined in the following way.

$$f(R) = \phi(H_2(R.x)),$$

where H_2 maps $\mathbb{Z}_p$ to $\{0,1\}^b$, $b < \lceil \log q \rceil$, and ϕ maps $\{0,1\}^b$ to $\mathbb{Z}_q^*$. The function H_2 can be instantiated by the function $H(1\|\overline{x}) \mod 2^b$, where H is the hash function that maps $\{0,1\}^*$ to $\mathbb{Z}_{2^{256}}$. The function ϕ is defined as follows: it maps $x \in \{0,1\}^b \setminus \{0\}$ to $\mathsf{int}(x)$ and maps zero to 2^b. This definition of the ϕ function is always correct due to the fact that $b < \lceil \log q \rceil$.

In order to separate domains of the hash function used for different cases, we redefine the hash function used for hashing messages as $H_1(x) = H(0\|\overline{x}) \mod q$.

We illustrate our method applying it to the GOST scheme which is a special case of the GenEG scheme. The formal definition of the GOST scheme is relegated to Appendix A. Note that function f in the GOST scheme (as well as in the ECDSA scheme) is defined as

$$f(R) = R.x \mod q.$$

We define the $\mathsf{GOST\text{-}H}$ scheme as follows.

```
KGen( )
  1 : $d \overset{\mathcal{U}}{\leftarrow} \mathbb{Z}_q^*$
  2 : $Q \leftarrow dP$
  3 : return $(d, Q)$
```

```
Sig(d, m)
  1 : $e \leftarrow H_1(m)$
  2 : if $e = 0 : e \leftarrow 1$
  3 : $k \overset{\mathcal{U}}{\leftarrow} \mathbb{Z}_q$
  4 : if $k = 0$ : return $\perp$
  5 : $R \leftarrow kP$
  6 : $r \leftarrow \phi(H_2(R.x))$
  7 : $s \leftarrow ke + dr$
  8 : if $s = 0$ : return $\perp$
  9 : return $\overline{r}\|\overline{s}$
```

```
Vf(Q, m, r̄‖s̄)
  1 : if $s = 0$ : return 0
  2 : $e \leftarrow H_1(m)$
  3 : if $e = 0 : e \leftarrow 1$
  4 : $R \leftarrow e^{-1}sP - e^{-1}rQ$
  5 : if $\phi(H_2(R.x)) \neq r$ : return 0
  6 : return 1
```

This method allows us to shorten the size of the signature from $(2\lceil \log q \rceil)$ bits to at most $(\lceil \log q \rceil + b + 1)$ bits. For instance, for $b = \lceil \log q \rceil / 2$ we can cut out one quarter of the size.

Note that we do not check the condition $r = 0$ in the $\mathsf{GOST\text{-}H}$ scheme, because the function ϕ is defined in a such way that it does not map any argument to zero.

3.2 $\mathsf{GenEG_S}$ scheme

The idea behind the second method is to «mine» during the signature generation procedure. We generate signature until it meets certain additional conditions: the first l bits of $\overline{r}$ should match the constant vector. Thus, we can exclude these bits from the signature and reduce the signature size by l bits.

The $\mathsf{GenEG_S.KGen}$ algorithm is similar to the $\mathsf{GenEG.KGen}$ algorithm. The $\mathsf{GenEG_S.Sig}$ and $\mathsf{GenEG_S.Vf}$ procedures are defined as follows.

```
Sig(d, m)
1 :  $cnt \leftarrow 0$
2 :  if $cnt > thr$ : return $\perp$
3 :  $cnt \leftarrow cnt + 1$
4 :  $\overline{r}\|\overline{s} \leftarrow \mathsf{GenEG.Sig}(d, m)$
5 :  if $\overline{r}\|\overline{s} = \perp$ : goto 2
6 :  if $\mathsf{lsb}_l(\overline{r}) \neq const$ : goto 2
7 :  $\overline{r}^* = \mathsf{msb}_{|\overline{r}|-l}(\overline{r})$
8 :  return $\overline{r}^*\|\overline{s}$
```

```
Vf(Q, m, r̄*‖s̄)
1 :  $\overline{r} \leftarrow \overline{r}^*\|const$
2 :  $res \leftarrow \mathsf{GenEG.Vf}(Q, m, \overline{r}\|\overline{s})$
3 :  return $res$
```

The scheme defined above has two new parameters: l – number of fixed bits in $\overline{r}$ and thr – number of attempts to generate valid signature. These parameters are not independent from each other and they are strictly related to the generation time and probability of outputting the valid signature by the $\mathsf{GenEG_S.Sig}$ procedure. Thus, they should be chosen in accordance with the generating mechanism computing power. We will discuss the appropriate values for these parameters in Section 5. The constant vector is an additional scheme parameter, it can be set to l zero bits for simplicity.

Note that the number of loop iterations needed to generate the valid signature depends on the probability of finding $\overline{r}$ satisfying the condition $\mathsf{lsb}_l(\overline{r}) = const$. In case of applying the method to the $\mathsf{GenEG\text{-}H}$ scheme, we can estimate this probability as 2^{-l} since the distribution of hash function output is close to uniform. We claim that the situation will not change in case of applying the method to schemes with function $f(R)$ equal to $R.x \mod q$ since function lsb_l is proven to be good entropy extractor for x-coordinate of point R (see [18] for more details).

3.3 $\mathsf{GenEG^V}$ scheme

The idea behind the third method is to truncate the signature (either $\overline{r}$ or $\overline{s}$ component) by t bits and search them during verification procedure.

The $\mathsf{GenEG}^{\mathsf{V}}.\mathsf{KGen}$ algorithm is similar to the $\mathsf{GenEG}.\mathsf{KGen}$ algorithm. The $\mathsf{GenEG}^{\mathsf{V}}.\mathsf{Sig}$ and $\mathsf{GenEG}^{\mathsf{V}}.\mathsf{Vf}$ procedures are defined as follows.

```
$\mathsf{Sig}(d, m)$
1 : $\overline{r}\|\overline{s} \leftarrow \mathsf{GenEG.Sig}(d, m)$
2 : if $\overline{r}\|\overline{s} = \perp$ : return $\perp$
3 : $\overline{s}^* \leftarrow \mathsf{msb}_{|\overline{s}|-t}(\overline{s})$
4 : return $\overline{r}\|\overline{s}^*$
```

```
$\mathsf{Vf}(Q, m, \overline{r}\|\overline{s}^*)$
1 : $i \leftarrow 0$
2 : if $i \geq 2^t$ : return $0$
3 : $\overline{s} \leftarrow \overline{s}^*\|\mathsf{str}_t(i)$
4 : $i \leftarrow i + 1$
5 : $res \leftarrow \mathsf{GenEG.Vf}(Q, m, \overline{r}\|\overline{s})$
6 : if $res = 0$ : goto 2
7 : return 1
```

The construction defined above assumes truncating the $\overline{s}$ component of the signature, however we can define this scheme similarly up to truncating the $\overline{r}$ component. The decision which part of the signature should be truncated could depend on the possible optimization of verification process.

This method allows us to reduce the signature size by t bits. The value of parameter t is strictly related to the signature verification time and should be chosen in accordance with the verifier's computing power.

Note that the proposed method is general and can be applied not only to the GenEG signature scheme but also to any scheme with signature represented as a concatenation of two bit vectors. In particular, it can be applied to the $\mathsf{GenEG}_{\mathsf{S}}$ scheme by replacing the $\mathsf{GenEG.Sig}$ and $\mathsf{GenEG.Vf}$ calls with the corresponding $\mathsf{GenEG}_{\mathsf{S}}$ procedure calls.

4 Security notions

In this section we formally define basic security models used for signature schemes and the assumptions on primitives.

Definition 1. *For a signature scheme* SS

$$\mathsf{Adv}_{\mathsf{SS}}^{\text{SUF-CMA}}(\mathcal{A}) = \Pr\left[\mathbf{Exp}_{\mathsf{SS}}^{\text{SUF-CMA}}(\mathcal{A}) \to 1\right],$$

where the experiment $\mathbf{Exp}_{\mathsf{SS}}^{\text{SUF-CMA}}(\mathcal{A})$ *is defined in the following way:*

```
$\mathbf{Exp}_{\mathsf{SS}}^{\text{SUF-CMA}}(\mathcal{A})$
1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SS.KGen}(\,)$
2 : $\mathcal{L} \leftarrow \emptyset$
3 : $(m, sgn) \xleftarrow{\$} \mathcal{A}^{Sign}(\mathsf{pk})$
4 : if $(m, sgn) \in \mathcal{L}$ : return $0$
5 : $res \leftarrow \mathsf{SS.Vf}(\mathsf{pk}, m, sgn)$
6 : return $res$
```

```
Oracle $Sign(m)$
1 : $sgn \leftarrow \mathsf{SS.Sig}(\mathsf{sk}, m)$
2 : $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, sgn)\}$
3 : return $sgn$
```

We analyse the security of the $\mathsf{GOST\text{-}H}$ scheme assuming the function H_2 to be a random oracle. The random oracle model was introduced in [27] and is an idealized model that assumes the existence of a public random function H such that all parties can obtain $H(x)$ (for any desired input value x) only by interacting with an oracle computing H; parties cannot compute H (for any input) on their own. Using a random oracle is a common way to ease the cryptographic analysis by making it modular. However, one should always keep in mind that a random oracle cannot be instantiated by any real hash function and, therefore, one should use it very carefully, trying to interpret the obtained security results. We discuss the meaning of random oracle model for our proof in the next section.

Definition 2 (ECDLP problem).

$$\mathsf{Adv}_{\mathbb{G}}^{\mathrm{ECDLP}}(\mathcal{A}) = \Pr\left[Q \xleftarrow{\mathcal{U}} \langle P \rangle\, ;\; d \xleftarrow{\$} \mathcal{A}(Q, P) : dP = Q\right]$$

Similar to [16] for the family H_1 of hash fuctions we define signum-relative collision resistance property (see Definition 3) and signum-relative division resistance property (see Definition 4). Throughout the paper we consider implicitly keyed hash functions H_1: $\{0,1\}^* \mapsto \mathbb{Z}_q$ with initialization vector assumed to be an implicit key. The experiments of the up-coming security definitions should be understood as implicitly first picking a random initialization vector $IV \in \mathcal{IV}$ and giving it to the adversary.

Definition 3 (SCR property). *For the family of hash functions* H_1

$$\mathsf{Adv}_{\mathsf{H}_1}^{\mathrm{SCR}}(\mathcal{A}) = \Pr\left[(m_1, m_2) \xleftarrow{\$} \mathcal{A} : H_1(m_1) = \pm H_1(m_2) \wedge m_1 \neq m_2\right]$$

Definition 4 (SDR property). *For the family of hash functions* H_1

$$\mathsf{Adv}_{\mathsf{H}_1}^{\mathrm{SDR}}(\mathcal{A}) =$$
$$\Pr\left[\beta_1, \beta_2 \xleftarrow{\mathcal{U}} \{0,1\}^b; (m_1, \Gamma) \xleftarrow{\$} \mathcal{A}_1(\beta_1), m_2 \xleftarrow{\$} \mathcal{A}_2(\Gamma, \beta_2) : \frac{H_1(m_1)}{\phi(\beta_1)} = \pm\frac{H_1(m_2)}{\phi(\beta_2)}\right]$$

The SDR property is implied by the standard assumptions: zero resistance and signum-relative preimage resistance properties of H_1 (see Appendix B.4 for formal proof and definitions of these properties).

We estimate the advantages defined above based on the best known methods of solving the corresponding security tasks. For the ECDLP problem it is the Pollard's ρ-algorithm (see [28]), for the SCR notion it is the attack based on birthday paradox and for the SDR notion (implied by the preimage resistance) it is the exhaustive search. So for the group $\mathbb{G}$ and for the family of hash functions H_1 we assume that for any adversary $\mathcal{A}$ with the time complexity at most T

$$\mathsf{Adv}_{\mathbb{G}}^{\mathrm{ECDLP}}(\mathcal{A}) \approx \frac{T^2}{q}; \quad \mathsf{Adv}_{\mathsf{H}_1}^{\mathrm{SCR}}(\mathcal{A}) \approx \frac{T^2}{q}; \quad \mathsf{Adv}_{\mathsf{H}_1}^{\mathrm{SDR}}(\mathcal{A}) \approx \frac{T}{q}.$$

5 Security bounds

In this section we provide the security bounds for the schemes defined in Section 3.

For the GOST-H scheme we provide the reduction from the ECDLP problem. We claim that the proof for the other GenEG-H schemes can be obtained using the same technique.

Theorem 1. *Let $\mathcal{A}$ be an adversary with time complexity at most T in the* SUF-CMA *model for the* GOST-H *scheme, making at most Q_S queries to the Sign oracle and at most Q_O queries to the H_2 oracle. Then there exists an adversary $\mathcal{D}$ that solves the* ECDLP *problem for the used elliptic curve group $\mathbb{G}$, an adversary $\mathcal{C}$ that breaks the signum-relative collision resistant property of* H_1 *and an adversary $\mathcal{M}$ that breaks the signum-relative division resistant property of* H_1*, such that:*

$$\mathsf{Adv}_{\mathsf{GOST\text{-}H}}^{\text{SUF-CMA}}(\mathcal{A}) \leqslant \sqrt{(Q_O+2)\left(\mathsf{Adv}_{\mathsf{H}_1}^{\mathrm{SDR}}(\mathcal{M}) \cdot (Q_O+2) + \mathsf{Adv}_{\mathbb{G}}^{\mathrm{ECDLP}}(\mathcal{D})\right)} + \\ + \frac{Q_O+3}{2^b} + \mathsf{Adv}_{\mathsf{H}_1}^{\mathrm{SCR}}(\mathcal{C}) + \frac{(2Q_O+Q_S+1)Q_S}{q-1}.$$

Furthermore, the time complexity of $\mathcal{C}$ is at most $T + c((Q_S+3)T^V_{\mathsf{GOST\text{-}H}} + Q_O)$, the time complexities of $\mathcal{D}$ and $\mathcal{M}$ are at most $2T + 2c((Q_S+4)T^V_{\mathsf{GOST\text{-}H}} + 2Q_O + 4)$, where $T^V_{\mathsf{GOST\text{-}H}}$ is computational resources needed to verify one signature by the GOST-H.Vf *procedure, c is a constant that depends only on a model of computation and a method of encoding.*

Here Q_S and Q_0 should be interpreted as a maximum number of signatures known to the adversary and as a maximum number of the hash function calls made by the adversary with the computational resources T respectively. The Q_S value is set in accordance with application requirements and the Q_O value depends on computational model and resources T. Usually we set Q_O to $\left\lceil \frac{T}{T_H} \right\rceil$, where T_H is the resources needed to compute one hash value for one-block message in the chosen computational model. It is correct as soon as we assume sequential computational model, however we note that any parallel model of computations is equivalent to the corresponding sequential computational model with more resources [26].

Proof sketch. The idea behind the proof is similar to the idea used in [15]. The proof consists of two steps. During the first one we show that the notions of existential unforgeability under chosen message attack and under key-only attack are nearly equivalent, assuming H_1 is signum-relative collision resistant. Next, using the forking lemma we show that the hardness of the ECDLP in the group $\mathbb{G}$ and the signum-relative division resistance of H_1 imply unforgeability under key-only attack. The full proof can be found in Appendix B.

Note that for several steps we provide more accurate reductions than article [15] does (there are several unclarified places which seem to be incorrect, for details see Appendix B). Note that providing accurate reductions is quite important since potential mistakes can lead to practical vulnerabilities (see e.g. [23, 24]).

The interpretation of the random oracle model in our case is as follows. If the signature scheme turns out to be insecure, then, due to the proof sketch, the ECDLP problem is solved or the used hash function does not sufficiently disrupt the link between the domain and the range.

Remark 1. *Note that the obtained reduction is not tight: there are no known cryptanalytic attacks breaking the signature scheme with the specified probability and computational resources. This is the common problem of reductions obtained using forking lemma. Moreover, several negative results are known. Paillier and Vergnaud [25] show that the forgeability of several discrete log based signatures cannot be equivalent to solving the discrete log problem in the standard model, assuming the so-called one-more discrete log assumption and algebraic reductions.*

The only term depening on b is $\frac{Q_O+3}{2^b}$. Applying the assumed bounds for $\mathsf{Adv}^{\mathrm{SDR}}_{\mathsf{H}_1}$, $\mathsf{Adv}^{\mathrm{SCR}}_{\mathsf{H}_1}$ and $\mathsf{Adv}^{\mathrm{ECDLP}}_{\mathbb{G}}$ we have the biggest term in the bound of order $\frac{\sqrt{Q_O}\cdot T}{\sqrt{q}}$. Thus, assuming $T > \sqrt{Q_O}$ (that is reasonable due to $Q_O \leqslant T$) we obtain the following surprising result: reducing b up to $\frac{\lceil \log_2 q \rceil}{2}$ does not significantly change the final bound. Thus, this method allows to shorten the size of the signature from $2\lceil \log_2 q \rceil$ bits to $\frac{3}{2}\lceil \log_2 q \rceil$ bits without harming security.

Theorem 2. *Let $\mathcal{A}$ be an adversary with time complexity at most T in the* SUF-CMA *model for the* $\mathsf{GenEG_S}$ *scheme, making at most Q_S queries to the Sign oracle. Then there exists an adversary $\mathcal{B}$ for the* GenEG *scheme in the* SUF-CMA *model that makes at most $Q_S \cdot thr$ queries to the Sign oracle, such that:*

$$\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{GenEG_S}}(\mathcal{A}) = \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{GenEG}}(\mathcal{B}).$$

Furthermore, the time complexity of $\mathcal{B}$ is at most $T + cQ_S thr$, where c is a constant that depends only on a model of computation and a method of encoding.

The proof can be found in Appendix C.

Consider the Q_S parameter in detail. Unlike the previous theorem, here Q_S cannot be interpreted as a number of signatures known to the adversary, since the scheme can return the failure indicator very often (depending on the parameters l and thr). Therefore, if N is a required for application number of signatures, then Q_S should be set to $\dfrac{N}{pr}$, where pr is the probability to return valid signature for one signing call. We assume that $pr \approx 1 - (1 - 2^{-l})^{thr}$.

Note that thr parameter should be chosen in such a way that the error probability is small enough for practice. The optimal way is to choose $thr = 2^l$ (in this case the error probability is less than e^{-1} for all l).

Theorem 3. *Let $\mathcal{A}$ be an adversary with time complexity at most T in the* SUF-CMA *model for the* $\mathsf{GenEG^V}$ *scheme, making at most Q_S queries to the Sign oracle. Then there exists an adversary $\mathcal{B}$ for the* GenEG *scheme in the* SUF-CMA *model that makes at most Q_S queries to the Sign oracle, such that:*

$$\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{GenEG^V}}(\mathcal{A}) = \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{GenEG}}(\mathcal{B}).$$

Furthermore, the time complexity of $\mathcal{B}$ is at most $T + c \cdot 2^t \cdot T^V_{\mathsf{GenEG}}$, where T^V_{GenEG} is computational resources needed to verify one signature by the $\mathsf{GenEG.Vf}$ *procedure, c is a constant that depends only on a model of computation and a method of encoding.*

The proof can be found in Appendix D.

Note that parameter t affects the security bound via the time complexity of the adversary $\mathcal{B}$. If we consider this parameter to be very large then the computational resources of $\mathcal{B}$ become too large, the GenEG scheme breaks and, as a result, the security bound degenerates.

The Theorems 2 and 3 are presented not in the random oracle model. However, if the security bound for the GenEG scheme is provided in the random oracle model we can change the theorems accordingly. If the adversary

$\mathcal{A}$ makes at most Q_O queries to H_2 oracle, then in case of the Theorem 2 the adversary $\mathcal{B}$ makes the same number of queries to its own H_2 oracle and in case of the Theorem 3 the adversary $\mathcal{B}$ makes at most $Q_O + 2^t$ queries to its own H_2 oracle.

The $\mathsf{GOST\text{-}H_S^V}$ scheme. Let us introduce the $\mathsf{GOST\text{-}H_S^V}$ scheme – the result of applying all three methods to the GOST scheme which is the special case of the GenEG scheme. The $\mathsf{GOST\text{-}H_S^V.KGen}$ algorithm is similar to the $\mathsf{GOST.KGen}$ algorithm. The $\mathsf{GOST\text{-}H_S^V.Sig}$ and $\mathsf{GOST\text{-}H_S^V.Vf}$ procedures are defined as follows.

```
Sig(d, m)
1 : $cnt \leftarrow 0$
2 : if $cnt > thr$ : return $\perp$
3 : $cnt \leftarrow cnt + 1$
4 : $\overline{r}\|\overline{s} \leftarrow \mathsf{GOST\text{-}H.Sig}(d, m)$
5 : if $\overline{r}\|\overline{s} = \perp$ : goto 2
6 : if $\mathsf{lsb}_l(\overline{r}) \neq const$ : goto 2
7 : $\overline{r}^* \leftarrow \mathsf{msb}_{|\overline{r}|-l}(\overline{r})$
8 : $\overline{s}^* \leftarrow \mathsf{msb}_{|\overline{s}|-t}(\overline{s})$
9 : return $\overline{r}^*\|\overline{s}^*$
```

```
Vf(Q, m, r̄*‖s̄*)
1 : $\overline{r} \leftarrow \overline{r}^*\|const$
2 : $i \leftarrow 0$
3 : if $i \geq 2^t$ : return 0
4 : $i \leftarrow i + 1$
5 : $\overline{s} \leftarrow \overline{s}^*\|\mathsf{str}_t(i)$
6 : $res \leftarrow \mathsf{GOST\text{-}H.Vf}(Q, m, \overline{r}\|\overline{s})$
7 : if $res = 0$ : goto 2
8 : return 1
```

Summarizing the results of Theorems 1, 2, 3 and the bounds presented in Section 4 we obtain the following security bound for the $\mathsf{GOST\text{-}H_S^V}$ scheme:

$$\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{GOST\text{-}H_S^V}}(\mathcal{A}) \leqslant \sqrt{2(Q_O + 2^t + 2) \cdot \frac{(Q_O + 2^t + 2)T_1 + 2T_1^2}{q}} + \\ + \frac{Q_O + 2^t + 3}{2^b} + \frac{(2Q_O + 2^{t+1} + Q_S \cdot thr + 1)Q_S \cdot thr + T_1^2}{q - 1},$$

where

$$T_1 \leqslant T + 2T^V_{\mathsf{GOST\text{-}H}}(Q_S \cdot thr + 2^t + 2) + 2Q_O + 4,$$

$T^V_{\mathsf{GOST\text{-}H}}$ is computational resources needed to verify one signature by the $\mathsf{GOST\text{-}H.Vf}$ procedure.

The size of the short signature generated by the $\mathsf{GOST\text{-}H_S^V}$ scheme is equal to at most $(\lceil \log q \rceil + b + 1 - l - t)$, where parameters b, l and t characterize three methods of shortening respectively. We provide the bounds for the $\mathsf{GOST\text{-}H_S^V}$ scheme for particular values of N, q, thr, T, Q_O: we set N to 10^6 as it is reasonable number of signatures for our application, we use curve with prime subgroup order q such that $2^{255} < q < 2^{256}$, we set thr to 2^l by reasons

Fixed parameters	Variable parameter and corresponding security bound					
$l = 18,\ t = 18$	b	128	100	80	70	60
	$\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{GOST\text{-}H}^{\mathsf{V}}_{\mathsf{S}}}(\mathcal{A})$	2^{-35}	2^{-35}	2^{-19}	2^{-9}	1
$b = 100,\ t = 18$	l	10	18	35	50	77
	$\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{GOST\text{-}H}^{\mathsf{V}}_{\mathsf{S}}}(\mathcal{A})$	2^{-35}	2^{-35}	2^{-34}	2^{-19}	1
$b = 100,\ l = 18$	t	10	18	30	64	80
	$\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{GOST\text{-}H}^{\mathsf{V}}_{\mathsf{S}}}(\mathcal{A})$	2^{-35}	2^{-35}	2^{-35}	2^{-27}	1

Table 1: Security bounds for the $\mathsf{GOST\text{-}H}^{\mathsf{V}}_{\mathsf{S}}$ scheme

discussed above and we set T to 2^{60} assuming such computational power of potential adversary for our application. Moreover, for simplicity we estimate Q_O as T. The $\mathsf{GOST\text{-}H.Vf}$ procedure assumes two hash and two multiple point calculation, we estimate $T^{V}_{\mathsf{GOST\text{-}H}}$ as 32 assuming the computational resources measured in hash calculations.

Table 1 presents the evolution of security bound with changing one of the scheme parameter as long as other two parameters are fixed. We choose l and t equal to 18 in the first step based on the appropriate signing (≈ 6 seconds) and verifying (≈ 3 seconds) time. The computer with the following characteristics was used: Intel Core i5-8600K CPU 3.60GHz, L1 D-Cache 32 KB x 6, L1 I-Cache 32 KB x 6, L2 Cache 256 KB x 6. We set b to 100 based on the security bound obtained in the first step.

By choosing the optimal values of methods parameters ($b = 100$, $l = 18$ and $t = 18$) we reduce the signature size from 512 to 320 bits providing the sufficient security for our application.

6 Conclusion

This paper introduces three methods of shortening ElGamal-type signatures. The proposed methods do not imply increasing the key sizes and can be applied together in any combination. Applying second and/or third method leads to increasing signing and/or verifying time. The implementation of these methods do not require any special cryptographic tools.

We apply these methods to ElGamal-type signature schemes and obtain security bounds in the random oracle model. The presented theorems allow us to estimate the security of the modified scheme by the security of the used cryptographic primitives (elliptic curve group and hash function family) in case of the first method and by the security of the original scheme in case of the second and the third methods. The paper presents the security bounds for the $\mathsf{GOST\text{-}H}_{\mathsf{S}}^{\mathsf{V}}$ scheme with elliptic curve subgroup order q, $2^{255} < q < 2^{256}$ with different parameter values (see Table 1). Choosing the optimal parameter values for our application allows to reduce the signature size from 512 to 320 bits.

References

[1] Rescorla, E., *The Transport Layer Security (TLS) Protocol Version 1.3*, RFC 8446, DOI 10.17487/RFC8446, 2018, https://www.rfc-editor.org/info/rfc8446.

[2] Kaufman, C., Hoffman P., Nir Y., Eronen P., Kivinen T., *Internet Key Exchange Protocol Version 2 (IKEv2)*, RFC 7296, DOI 10.17487/RFC7296, 2014, https://www.rfc-editor.org/info/rfc7296.

[3] Boneh D., Lynn B., Shacham H., “Short Signatures from the Weil Pairing”, *LNCS*, Advances in Cryptology — ASIACRYPT 2001, **2248**, ed. Boyd C., Springer, Berlin, Heidelberg, 2001.

[4] Patarin J., Courtois N., Goubin L., “FLASH, a Fast Multivariate Signature Algorithm”, *LNCS*, Topics in Cryptology — CT-RSA 2001, **2020**, ed. Naccache D., Springer, Berlin, Heidelberg, 2001.

[5] Patarin J., Courtois N., Goubin L., “QUARTZ, 128-Bit Long Digital Signatures”, *LNCS*, Topics in Cryptology — CT-RSA 2001, **2020**, ed. Naccache D., Springer, Berlin, Heidelberg, 2001.

[6] Dubois V., Fouque PA., Shamir A., Stern J., “Practical Cryptanalysis of SFLASH”, *LNCS*, Advances in Cryptology - CRYPTO 2007, **4622**, ed. Menezes A., Springer, Berlin, Heidelberg, 2007.

[7] Courtois N.T., Daum M., Felke P., “On the Security of HFE, HFEv- and Quartz”, *LNCS*, Public Key Cryptography — PKC 2003, **2567**, ed. Desmedt Y.G., Springer, Berlin, Heidelberg, 2003.

[8] Courtois N.T., Finiasz M., Sendrier N., “How to Achieve a McEliece-Based Digital Signature Scheme”, *LNCS*, Advances in Cryptology — ASIACRYPT 2001, **2248**, ed. Boyd C., Springer, Berlin, Heidelberg, 2001.

[9] Koblitz N., “Hidden Monomial Cryptosystems”, *Algebraic Aspects of Cryptography, Algorithms and Computation in Mathematics*, **3**, Springer, Berlin, Heidelberg, 1998, 80–102.

[10] Petzoldt A., Chen MS., Yang BY., Tao C., Ding J., “Design Principles for HFEv- Based Multivariate Signature Schemes”, *LNCS*, Advances in Cryptology – ASIACRYPT 2015, **9452**, ed. Iwata T., Cheon J., Springer, Berlin, Heidelberg, 2015.

[11] Mohamed M.S.E., Petzoldt A., “The Shortest Signatures Ever”, *LNCS*, Progress in Cryptology – INDOCRYPT 2016, **10095**, ed. Dunkelman O., Sanadhya S., Springer, Cham, 2016.

[12] Kipnis A., Patarin J., Goubin L., “Unbalanced Oil and Vinegar Signature Schemes”, *LNCS*, Advances in Cryptology — EUROCRYPT’99, **1592**, ed. Stern J., Springer, Berlin, Heidelberg, 1999.

[13] Ding J., Schmidt D., “Rainbow, a New Multivariable Polynomial Signature Scheme”, *LNCS*, Applied Cryptography and Network Security. ACNS 2005, **3531**, ed. Ioannidis J., Keromytis A., Yung M., Springer, Berlin, Heidelberg, 2005.

[14] Fersch, M., Kiltz, E., Poettering, B., “On the One-Per-Message Unforgeability of (EC)DSA and Its Variants”, *LNCS*, Theory of Cryptography. TCC 2017, **10678**, ed. Kalai Y., Reyzin L., Springer, Cham, 2017.

[15] Fersch M., Kiltz E., Poettering B., “On the provable security of (EC) DSA signatures”, Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, 1651–1662.

[16] Fersch, M., *The provable security of Elgamal-type signature schemes*, Diss. Bochum, Ruhr-Universität Bochum, 2018.

[17] Bellare M., Rogaway P., "The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs", *LNCS*, Advances in Cryptology - EUROCRYPT 2006, **4004**, ed. Vaudenay S., Springer, Berlin, Heidelberg, 2006.

[18] Chevalier C., Fouque PA., Pointcheval D., Zimmer S., "Optimal Randomness Extraction from a Diffie-Hellman Element", *LNCS*, Advances in Cryptology - EUROCRYPT 2009, **5479**, ed. Joux A., Springer, Berlin, Heidelberg, 2009.

[19] *GOST R 34.10-2012. Information technology. Cryptographic data security. Signature and verification processes of electronic digital signature. National standard of the Russian Federation, STANDARTINFORM*, 2012, In Russian.

[20] *GOST 34.10-2018. Information technology. Cryptographic data security. Signature and verification processes of electronic digital signature. Interstate standard, Interstate Council for Standardization, Metrology and Certification (ISC)*, 2018, In Russian.

[21] *ISO/IEC 14888-3:2018, IT Security techniques – Digital signatures with appendix – Part 3: Discrete logarithm based mechanisms – Section 6: Certificate-based mechanisms – 6.9: ECRDSA*, 2018.

[22] Dolmatov V., Degtyarev A., *GOST R 34.10-2012: Digital Signature Algorithm*, RFC 7091, DOI 10.17487/RFC7091, 2013, https://www.rfc-editor.org/info/rfc7091.

[23] Inoue A., Iwata T., Minematsu K., Poettering B., "Cryptanalysis of OCB2: Attacks on Authenticity and Confidentiality", *LNCS*, Advances in Cryptology – CRYPTO 2019, **11692**, ed. Boldyreva A., Micciancio D., Springer, Berlin, Heidelberg, 2019.

[24] Koblitz N., Menezes A., *Critical Perspectives on Provable Security: Fifteen Years of "Another Look" Papers*, Cryptology ePrint Archive: Report 2019/1336, 2019.

[25] Paillier P., Vergnaud D., "Discrete-Log-Based Signatures May Not Be Equivalent to Discrete Log", *LNCS*, Advances in Cryptology - ASIACRYPT 2005, **3788**, ed. Roy B., Springer, Berlin, Heidelberg, 2005.

[26] Savage J.E., *Models of Computation: Exploring the Power of Computing*, Addison-Wesley Longman Publishing Co, Boston, 1998.

[27] Bellare M., Rogaway P., "Random oracles are practical: A paradigm for designing efficient protocols", *Proceedings of the 1st ACM conference on Computer and communications security*, 1993, 62–73.

[28] Pollard, J.M., "A monte carlo method for factorization", *BIT*, **15** (1975), 331–334.

[29] Zheng Y., "Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption)", *LNCS*, Advances in Cryptology — CRYPTO'97, **1294**, ed. Kaliski B.S., Springer, Berlin, Heidelberg, 1997.

A GOST definition

The GOST signature scheme is a special case of the GenEG scheme. We define it relative to functions H, f and group $\mathbb{G}$.

```
KGen( )
1 : $d \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
2 : $Q \leftarrow dP$
3 : return $(d, Q)$
```

```
Sig$(d, m)$
1 : $e \leftarrow H(m) \mod q$
2 : if $e = 0 : e \leftarrow 1$
3 : $k \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
4 : if $k = 0$ : return $\perp$
5 : $R \leftarrow kP$
6 : $r \leftarrow f(R)$
7 : if $r = 0$ : return $\perp$
8 : $s \leftarrow ke + dr$
9 : if $s = 0$ : return $\perp$
10 : return $\overline{r}\|\overline{s}$
```

```
Vf$(Q, m, \overline{r}\|\overline{s})$
1 : if $(r = 0 \vee s = 0)$ : return 0
2 : $e \leftarrow H(m)$
3 : if $e = 0 : e \leftarrow 1$
4 : $R \leftarrow e^{-1}sP - e^{-1}rQ$
5 : if $f(R) \neq r$ : return 0
6 : return 1
```

The function f maps $\mathbb{G}^*$ to $\mathbb{Z}_q$ and for the GOST scheme is defined as follows:

$$f(R) = R.x \mod q.$$

There are some differences between the scheme defined above and the standardized scheme defined in [19]. First, our version of the scheme can output failure indicator $\perp$. In contrast, function Sig in standardized scheme always output a valid signature, going to line 3 in case of all «bad» events. The second difference is that k is chosen randomly from the set $\mathbb{Z}_q^*$ in the standardized scheme, but in our scheme it is chosen from the set $\mathbb{Z}_q$, and the procedure Sig outputs $\perp$ in case of $k = 0$. We claim that these two differences do not affect the security and correctness of the scheme.

B GOST-H security

B.1 Proof details

We provide the proof in the random oracle model, i.e. we replace function H_2 with the random oracle. Family of hash functions H_1 is required to be signum-relative collision resistant and signum-relative division resistant in the sense of Definitions 3, 4.

Let us introduce SUF-KO security model for the signature scheme since we use it during the proof.

Definition 5. *For a signature scheme* SS

$$\mathsf{Adv}^{\text{SUF-KO}}_{\mathsf{SS}}(\mathcal{A}) = \Pr\left[\mathbf{Exp}^{\text{SUF-KO}}_{\mathsf{SS}}(\mathcal{A}) \to 1\right],$$

where the experiment $\mathbf{Exp}^{\text{SUF-KO}}_{\mathsf{SS}}(\mathcal{A})$ *is defined in the following way:*

```
$\mathbf{Exp}^{\text{SUF-KO}}_{\mathsf{SS}}(\mathcal{A})$
1 :  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SS.KGen}(\,)$
2 :  $(m, sgn) \overset{\$}{\leftarrow} \mathcal{A}(\mathsf{pk})$
3 :  $res \leftarrow \mathsf{SS.Vf}(\mathsf{pk}, m, sgn)$
4 :  return $res$
```

The idea behind the proof is similar to the idea used in [15], however several steps in [15] are unclear and seem to be incorrect. We provide more accurate reduction than [15] does and point out the differences from [15] throughout the proof. We split the proof into two parts. In Section B.2 we show that if the adversary can forge the GOST-H scheme using some valid pairs message-signature (i.e. in the SUF-CMA model), then we can construct two adversaries: one of them breaks the signum-relative collision resistance property of H_1 and the other one makes forgery without any valid pairs message-signature (i.e. in the SUF-KO model). In Section B.3 we construct the adversary that breaks the signum-relative division resistance property of H_1 and the adversary that solves the ECDLP problem using the key-only adversary constructed at the first step. The forking lemma (see [15]) is our key tool on the second step. Both steps of the proof are organized as follows: at first, we construct the sequence of experiments for the adversary and estimate the difference between them (in some cases by constructing the adversaries for H_1 properties), after that we build another adversary who uses the first adversary as a black box and implements the last experiment for him. We highlight the changes in the experiment pseudocode.

We write **abort** in the experiment pseudocode as a shortcut for «**return** 0» and in the oracle pseudocode to denote that experiment should stop and return 0. We use lemma 2 from [17] to estimate the difference between two experiments $\mathbf{Exp}^{i}$ and $\mathbf{Exp}^{j}$ that are «identical-until-bad», i.e. one experiment is derived from the other by adding the abort condition. According to this lemma

$$\Pr\left[\mathbf{Exp}^{i} \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^{j} \Rightarrow 1\right] \leqslant \Pr[\text{abort condition is met}].$$

For presentation purposes we introduce the internal result of function f and denote it as r'. More particularly, we denote $\mathsf{H}_2(R.x)$ as r' and thus $r = \phi(r')$. We assume that r' is the element in $\{0,1\}^b$. Moreover, we assume throughout the proof that the GenEG signature is represented not like the vector's concatenation but as the pair of corresponding elements in $\mathbb{Z}_q$.

B.2 SUF-KO to SUF-CMA reduction

Theorem 4. *Let $\mathcal{A}$ be an adversary with time complexity at most T in the* SUF-CMA *model for the* GOST-H *scheme, making at most Q_S queries to the Sign oracle and at most Q_O queries to the H_2 oracle. Then there exists an adversary $\mathcal{B}$ in the* SUF-KO *model for the* GOST-H *scheme making at most (Q_O+2) queries to the H_2 oracle and exists an adversary $\mathcal{C}$ that breaks the signum-relative collision resistant property of* H_1*, such that:*

$$\mathsf{Adv}^{\text{SUF-CMA}}_{\text{GOST-H}}(\mathcal{A}) \leqslant \mathsf{Adv}^{\text{SUF-KO}}_{\text{GOST-H}}(\mathcal{B}) + \mathsf{Adv}^{\text{SCR}}_{\mathsf{H}_1}(\mathcal{C}) + \frac{(2Q_O+Q_S+1)Q_S}{q-1}.$$

Furthermore, the time complexities of $\mathcal{B}$ and $\mathcal{C}$ are at most $T + c((Q_S+3)T^V_{\text{GOST-H}} + Q_O)$, where $T^V_{\text{GOST-H}}$ is computational resources needed to verify one signature by the GOST-H.Vf *procedure, c is a constant that depends only on a model of computation and a method of encoding.*

Construction of adversary $\mathcal{C}$. Let $\mathbf{Exp}^0$ denote the original security experiment as defined in the SUF-CMA security model definition (see Figure 1). We fix $\mathcal{A}$ – the adversary that makes forgery for the GOST-H scheme in the SUF-CMA model. The adversary has the access to the random oracle H_2 and to the signing oracle *Sign*. We assume that adversary can make at most Q_O queries to the oracle H_2 and Q_S queries to the oracle *Sign*. Our goal is to upper-bound $\Pr\left[\mathbf{Exp}^{\text{SUF-CMA}}_{\text{GOST-H}}(\mathcal{A}) \Rightarrow 1\right] = \Pr\left[\mathbf{Exp}^0(\mathcal{A}) \Rightarrow 1\right]$.

Note that we change the check for k being equal to zero (see line 4 in the GOST-H.Sig procedure, Section 3.1) to the check for R being equal to zero point (see line 5 in the *Sign* oracle). This change does not affect the scheme but simplifies the proof.

$\mathbf{Exp}^1$ is the modification of the $\mathbf{Exp}^0$ obtained by implementing H_2 using «lazy sampling» (see Figure 2). The idea is to «open» new pairs $(x, H_2(x))$ as soon as the adversary asks for it. We introduce the set Π – the subset of $(\mathbb{Z}_p, \{0,1\}^b)$, which is defined by the union of two sets Π^S and Π^O. We store the pairs obtained from queries to the H_2 oracle in Π^O set and the pairs obtained from queries to the *Sign* oracle in Π^S set. If $(\alpha,\beta) \in \Pi$, we denote β as $\Pi(\alpha)$. We write $(\alpha,\cdot) \in \Pi$ shorthand for the condition that there exists β such that $(\alpha,\beta) \in \Pi$.

This modification does not affect the distribution of H_2 and *Sign* outputs. Thus, $\Pr\left[\mathbf{Exp}^0(\mathcal{A}) \Rightarrow 1\right] = \Pr\left[\mathbf{Exp}^1(\mathcal{A}) \Rightarrow 1\right]$.

Proof.

```
$\mathbf{Exp}^0(\mathcal{A}) = \mathbf{Exp}^{\text{SUF-CMA}}_{\textsf{GOST-H}}(\mathcal{A})$
1 :  $d \overset{\mathcal{U}}{\leftarrow} \mathbb{Z}_q^*$
2 :  $Q \leftarrow dP$
3 :  $H_2 \overset{\mathcal{U}}{\leftarrow} Func(\mathbb{Z}_p, \{0,1\}^b)$
4 :  $\mathcal{L} \leftarrow \emptyset$
........Setup completed........
5 :  $(m, \langle r, s\rangle) \overset{\$}{\leftarrow} \mathcal{A}^{Sign, H_2}(Q)$
6 :  if $(m, \langle r, s\rangle) \in \mathcal{L}$ : abort
7 :  if $s = 0$ : abort
8 :  $e \leftarrow H_1(m)$
9 :  if $e = 0 : e \leftarrow 1$
10 : $R \leftarrow e^{-1}sP - e^{-1}rQ$
11 : if $\phi(H_2(R.x)) \neq r$ : abort
12 : return 1
```

```
Oracle $Sign(m)$
1 :  $e \leftarrow H_1(m)$
2 :  if $e = 0 : e \leftarrow 1$
3 :  $k \overset{\mathcal{U}}{\leftarrow} \mathbb{Z}_q$
4 :  $R \leftarrow kP$
5 :  if $R = 0$ : return $\perp$
6 :  $r' \leftarrow H_2(R.x)$
7 :  $r \leftarrow \phi(r')$
8 :  $s \leftarrow ke + dr$
9 :  if $s = 0$ : return $\perp$
10 : $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, \langle r, s\rangle)\}$
11 : return $\langle r, s\rangle$
```

```
Oracle $H_2(\alpha)$
1 :  return $H_2(\alpha)$
```

Figure 1: The $\mathbf{Exp}^0$ for the adversary $\mathcal{A}$ for the GOST-H scheme in the SUF-CMA model

$\mathbf{Exp}^2$ is the modification of the $\mathbf{Exp}^1$ in which forgeries obtained by finding a signum-relative collision are not counted (see Figure 2, lines 7, 8, 9 are added). The *Sign* and H_2 oracles do not change from the $\mathbf{Exp}^1$.

To estimate the difference between the $\mathbf{Exp}^1$ and $\mathbf{Exp}^2$, we should estimate the probability that the $\mathbf{Exp}^2$ aborts in line 9.

Let construct an adversary $\mathcal{C}$ that breaks the signum-relative collision resistant property of H_1. The adversary $\mathcal{C}$ implements the $\mathbf{Exp}^2$ for $\mathcal{A}$. Note that he is able to do this as soon as we replace H_2 implementation with lazy sampling. Otherwise, the polynomial-time bounded adversary could not choose function H_2 randomly from the set $Func(\mathbb{Z}_p, \{0,1\}^b)$ cause the density of this set is exponential. $\mathcal{A}$ delivers a forgery to $\mathcal{C}$, and $\mathcal{C}$ finds the signum-relative collision iff the condition in lines 7-8 is met.

Thus, we obtain the following bound:

$$\Pr\left[\mathbf{Exp}^1(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^2(\mathcal{A}) \Rightarrow 1\right] \leqslant \mathsf{Adv}^{\text{SCR}}_{\mathsf{H}_1}(\mathcal{C}).$$

The adversary $\mathcal{C}$ implements $\mathbf{Exp}^2$ and thus processes at most Q_S queries to *Sign* oracle and at most Q_O queries to H_2 oracle, checks the collision condition and verifies the forgery obtained from $\mathcal{A}$. Taking into account that signature generation procedure and hash computation are faster than verification procedure, $\mathcal{C}$ uses at most $c((Q_S+2)T^V_{\textsf{GOST-H}} + Q_O)$ additional computational resources, where $T^V_{\textsf{GOST-H}}$ is computational resources needed to verify

Exp$^1(\mathcal{A})$

```
1 :  $d \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
2 :  $Q \leftarrow dP$
3 :  $(\Pi^O, \Pi^S) \leftarrow (\emptyset, \emptyset)$
4 :  $\Pi \leftarrow \Pi^O \cup \Pi^S$
5 :  $\mathcal{L} \leftarrow \emptyset$
........Setup completed........
6 :  $(m, \langle r, s \rangle) \xleftarrow{\$} \mathcal{A}^{Sign, H_2}(Q)$
7 :  $\textbf{if } (m, \langle r, s \rangle) \in \mathcal{L} : \textbf{abort}$
8 :  $\textbf{if } s = 0 : \textbf{abort}$
9 :  $e \leftarrow H_1(m)$
10 : $\textbf{if } e = 0 : e \leftarrow 1$
11 : $R \leftarrow e^{-1}sP - e^{-1}rQ$
12 : $\textbf{if } \phi(H_2(R.x)) \neq r : \textbf{abort}$
13 : $\textbf{return } 1$
```

Exp$^2(\mathcal{A})$

```
1 :  $d \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
2 :  $Q \leftarrow dP$
3 :  $(\Pi^O, \Pi^S) \leftarrow (\emptyset, \emptyset)$
4 :  $\Pi \leftarrow \Pi^O \cup \Pi^S$
5 :  $\mathcal{L} \leftarrow \emptyset$
........Setup completed........
6 :  $(m, \langle r, s \rangle) \xleftarrow{\$} \mathcal{A}^{Sign, H_2}(Q)$
7 :  $\forall (m^*, \cdot) \in \mathcal{L}, m^* \neq m :$
8 :    $\textbf{if } H_1(m^*) = \pm H_1(m) :$
9 :      $\textbf{abort}$
10 : $\textbf{if } (m, \langle r, s \rangle) \in \mathcal{L} : \textbf{abort}$
11 : $\textbf{if } s = 0 : \textbf{abort}$
12 : $e \leftarrow H_1(m)$
13 : $\textbf{if } e = 0 : e \leftarrow 1$
14 : $R \leftarrow e^{-1}sP - e^{-1}rQ$
15 : $\textbf{if } \phi(H_2(R.x)) \neq r : \textbf{abort}$
16 : $\textbf{return } 1$
```

Oracle $Sign(m)$

```
1 :  $e \leftarrow H_1(m)$
2 :  $\textbf{if } e = 0 : e \leftarrow 1$
3 :  $k \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
4 :  $R \leftarrow kP$
5 :  $\textbf{if } R = 0 : \textbf{return } \bot$
6 :  $\textbf{if } (R.x, \cdot) \in \Pi :$
7 :    $r' \leftarrow \Pi(R.x)$
8 :  $\textbf{else} :$
9 :    $r' \xleftarrow{\mathcal{U}} \{0,1\}^b$
10 :   $\Pi^S \leftarrow \Pi^S \cup \{(R.x, r')\}$
11 :   $\Pi \leftarrow \Pi^O \cup \Pi^S$
12 : $r \leftarrow \phi(r')$
13 : $s \leftarrow ke + dr$
14 : $\textbf{if } s = 0 : \textbf{return } \bot$
15 : $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, \langle r, s \rangle)\}$
16 : $\textbf{return } \langle r, s \rangle$
```

Oracle $H_2(\alpha)$

```
1 :  $\textbf{if } (\alpha, \cdot) \in \Pi :$
2 :    $\textbf{return } \Pi(\alpha)$
3 :  $\beta \xleftarrow{\mathcal{U}} \{0,1\}^b$
4 :  $\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
5 :  $\Pi \leftarrow \Pi^O \cup \Pi^S$
6 :  $\textbf{return } \beta$
```

Figure 2: The **Exp**^{1} and **Exp**^{2} for the adversary $\mathcal{A}$ for the GOST-H scheme in the SUF-CMA model. The *Sign* and H_2 oracles are the same in the **Exp**^{1} and **Exp**^{2}

one signature by the GOST-H.Vf procedure, c is a constant that depends only on a model of computation and a method of encoding.

Note that if we find signum-relative collision then we immediately construct a forgery for the GOST-H scheme (it is also true for the GOST scheme) in the SUF-CMA model. It is the interesting property of the GOST signature scheme implied by its construction, namely the equation for s component computation. The probability of such collision event is part of the resulting security bound.

Construction of adversary $\mathcal{B}$. In the further experiments we change the *Sign* oracle behaviour only (see Figure 3).

```
Oracle $Sign(m)$ ($\mathbf{Exp}^3$)
1 :  $e \leftarrow H_1(m)$
2 :  if $e = 0 : e \leftarrow 1$
3 :  $k \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
4 :  $R \leftarrow kP$
5 :  if $R = 0$ : return $\perp$
6 :  if $(R.x, \cdot) \in \Pi$ :
7 :    abort
8 :  $r' \xleftarrow{\mathcal{U}} \{0,1\}^b$
9 :  $\Pi^S \leftarrow \Pi^S \cup \{(R.x, r')\}$
10 : $\Pi \leftarrow \Pi^O \cup \Pi^S$
11 : $r \leftarrow \phi(r')$
12 : $s \leftarrow ke + dr$
13 : if $s = 0$ : return $\perp$
14 : $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, \langle r, s\rangle)\}$
15 : return $\langle r, s\rangle$
```

```
Oracle $Sign(m)$ ($\mathbf{Exp}^4$)
1 :  $e \leftarrow H_1(m)$
2 :  if $e = 0 : e \leftarrow 1$
3 :  $r' \xleftarrow{\mathcal{U}} \{0,1\}^b$
4 :  $r \leftarrow \phi(r')$
5 :  $s \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
6 :  $R \leftarrow e^{-1}sP - e^{-1}rQ$
7 :  if $R = 0$ : return $\perp$
8 :  if $(R.x, \cdot) \in \Pi$ :
9 :    abort
10 : $\Pi^S \leftarrow \Pi^S \cup \{(R.x, r')\}$
11 : $\Pi \leftarrow \Pi^O \cup \Pi^S$
12 : if $s = 0$ : return $\perp$
13 : $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, \langle r, s\rangle)\}$
14 : return $\langle r, s\rangle$
```

```
Oracle $Sign(m)$ ($\mathbf{Exp}^5$)
1 :  $e \leftarrow H_1(m)$
2 :  if $e = 0 : e \leftarrow 1$
3 :  $r' \xleftarrow{\mathcal{U}} \{0,1\}^b$
4 :  $r \leftarrow \phi(r')$
5 :  $s \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
6 :  if s = 0: abort
7 :  $R \leftarrow e^{-1}sP - e^{-1}rQ$
8 :  if $R = 0$ : return $\perp$
9 :  if $(R.x, \cdot) \in \Pi$ :
10 :   abort
11 : $\Pi^S \leftarrow \Pi^S \cup \{(R.x, r')\}$
12 : $\Pi \leftarrow \Pi^O \cup \Pi^S$
13 : if $s = 0$ : return $\perp$
14 : $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, \langle r, s\rangle)\}$
15 : return $\langle r, s\rangle$
```

Figure 3: The *Sign* oracles in the $\mathbf{Exp}^3$, $\mathbf{Exp}^4$, $\mathbf{Exp}^5$

The *Sign* oracle in the $\mathbf{Exp}^3$ is the modification of the *Sign* oracle in the $\mathbf{Exp}^2$ by adding the abort condition in case of choosing $R.x$ that already belongs to set Π (lines 6-7). We should estimate the probability of this event to estimate the difference between the $\mathbf{Exp}^2$ and $\mathbf{Exp}^3$.

The value k is uniformly distributed in a set $\mathbb{Z}_q^*$ of cardinality $(q-1)$. Thus, $R.x$ is uniformly distributed in a set of cardinality $(q-1)/2$. In the worst case the adversary $\mathcal{A}$ has already made all queries to the H_2 oracle and thus Π contains at least Q_O elements. The abort condition is met if the value $R.x$ hits one of elements in Π. We can estimate this probability as

$\dfrac{Q_O + Q_S/2}{(q-1)/2} = \dfrac{2Q_O + Q_S}{q-1}$. As these lines are executed at most Q_S times, the overall probability can be bounded by $\dfrac{(2Q_O + Q_S)Q_S}{q-1}$.

We obtain the following bound:

$$\Pr\left[\mathbf{Exp}^2(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^3(\mathcal{A}) \Rightarrow 1\right] \leqslant \frac{(2Q_O + Q_S)Q_S}{q-1}.$$

The signature oracle in the $\mathbf{Exp}^4$ gets along with only public information. Values r' and s are randomly chosen from the relevant sets and then point R is constructed. We define the corresponding pair in H_2 implementation by saving this pair in the Π^S set. Note that if we couldn't do so (i.e., $R.x$ already belongs to the Π), the abort condition is met like in the $\mathbf{Exp}^3$. This step differs from [15] in the order of $\perp$ outputs and **abort** conditions.

Consider the distribution on r', s and $\perp$. Note that if the distributions on r' are identical in both experiments then the distributions on r are identical too. In the $\mathbf{Exp}^3$ r' is distributed uniformly in $\{0,1\}^b$, k is distributed uniformly in $\mathbb{Z}_q^*$ except of the values that lead to $R.x$ that already belongs to Π. k and r are independent from each over in the equation for s, therefore s is distributed uniformly in $\mathbb{Z}_q$ except of the values corresponding to «bad» values of k. In the $\mathbf{Exp}^4$ r' and s are also distributed uniformly on the corresponding sets and s values that lead to the same «bad» events as in the $\mathbf{Exp}^3$ are excluded. The probability of returning $\perp$ is also the same in these experiments.

The probability of abort in the $\mathbf{Exp}^3$ and $\mathbf{Exp}^4$ is the same because R is uniformly distributed in the set of cardinality q in both experiments and zero point is excluded.

Thus, we conclude that

$$\Pr\left[\mathbf{Exp}^3(\mathcal{A}) \Rightarrow 1\right] = \Pr\left[\mathbf{Exp}^4(\mathcal{A}) \Rightarrow 1\right].$$

Finally we are moving to the $\mathbf{Exp}^5$. The abort condition in line 6 is added to the signing oracle. Note that line 13 is redundant now, however we keep it for clarity. The qualitative significance of this modification is following: the set Π^S contains only those pairs $(R.x, r')$ that lead to valid signatures now, because the condition in line 13 can never be met. Note that there is no such step in [15].

We estimate the difference between the $\mathbf{Exp}^4$ and $\mathbf{Exp}^5$ by estimating the probability of abort condition in line 6. Per each execution it is equal to $1/q$, so the overall probability can be bounded by Q_S/q.

We obtain the following bound:

$$\Pr\left[\mathbf{Exp}^4(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^5(\mathcal{A}) \Rightarrow 1\right] \leqslant \frac{Q_S}{q}.$$

Before constructing the adversary $\mathcal{B}$ we summarize the obtained bounds:

$$\begin{gathered}\Pr\left[\mathbf{Exp}^0(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^5(\mathcal{A}) \Rightarrow 1\right] = (\Pr\left[\mathbf{Exp}^0(\mathcal{A}) \Rightarrow 1\right] - \\ - \Pr\left[\mathbf{Exp}^1(\mathcal{A}) \Rightarrow 1\right]) + \left(\Pr\left[\mathbf{Exp}^1(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^2(\mathcal{A}) \Rightarrow 1\right]\right) + \\ + \left(\Pr\left[\mathbf{Exp}^2(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^3(\mathcal{A}) \Rightarrow 1\right]\right) + (\Pr\left[\mathbf{Exp}^3(\mathcal{A}) \Rightarrow 1\right] - \\ - \Pr\left[\mathbf{Exp}^4(\mathcal{A}) \Rightarrow 1\right]) + \left(\Pr\left[\mathbf{Exp}^4(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^5(\mathcal{A}) \Rightarrow 1\right]\right) \leqslant \\ \leqslant \mathsf{Adv}^{\mathrm{SCR}}_{\mathsf{H}_1}(\mathcal{C}) + \frac{(2Q_O + Q_S)Q_S}{q-1} + \frac{Q_S}{q} \leqslant \mathsf{Adv}^{\mathrm{SCR}}_{\mathsf{H}_1}(\mathcal{C}) + \frac{(2Q_O + Q_S + 1)Q_S}{q-1}.\end{gathered}$$

Let construct the adversary $\mathcal{B}$ for the GOST-H scheme in the SUF-KO model that uses $\mathcal{A}$ as the black box (see Figure 4).

```
$\mathcal{B}^{H_2^*}(Q)$

$(\Pi^O, \Pi^S) \leftarrow (\emptyset, \emptyset)$
$\Pi \leftarrow \Pi^O \cup \Pi^S$
$\mathcal{L} \leftarrow \emptyset$
$(m, \langle r, s\rangle) \xleftarrow{\$} \mathcal{A}^{SimSign, SimH_2}(Q)$
$\forall (m^*, \cdot) \in \mathcal{L}, m^* \neq m :$
    if $H_1(m^*) = \pm H_1(m) :$
        abort
if $(m, \langle r, s\rangle) \in \mathcal{L} :$ abort
if $s = 0 :$ abort
$e \leftarrow H_1(m)$
if $e = 0 : e \leftarrow 1$
$R \leftarrow e^{-1}sP - e^{-1}rQ$
if $\phi(SimH_2(R.x)) \neq r :$ abort
$r' \leftarrow \phi^{-1}(r)$
if $(R.x, r') \in \Pi^S :$
    Find corresponding $(m_1, \langle r_1, s_1\rangle) \in \mathcal{L}$
    Compute $d$
    $(m, \langle r, s\rangle) \xleftarrow{\$} Sign^{\mathcal{B}}(d, m)$
return $(m, \langle r, s\rangle)$
```

```
$SimH_2(\alpha)$

if $(\alpha, \cdot) \in \Pi :$
    return $\Pi(\alpha)$
$\beta \leftarrow H_2^*(\alpha)$
$\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
$\Pi \leftarrow \Pi^O \cup \Pi^S$
return $\beta$
```

```
$Sign^{\mathcal{B}}(d, m)$

$e \leftarrow H_1(m)$
if $e = 0 : e \leftarrow 1$
$k \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
$R \leftarrow kP$
$r \leftarrow \phi(H_2^*(R.x))$
$s \leftarrow ke + dr$
if $s = 0 :$
    $e_1 \leftarrow H_1(m_1)$
    if $e_1 = 0 : e_1 \leftarrow 1$
    $s_1 \leftarrow ke_1 + dr$
return $m_1, \langle r, s_1\rangle$
```

Figure 4: The adversary $\mathcal{B}$ for the GOST-H scheme in the SUF-KO model that uses the adversary $\mathcal{A}$ for the GOST-H scheme in the SUF-CMA model

Adversary $\mathcal{B}$ simulates the $Sign$ and H_2 oracles using $SimSign$ and $SimH_2$ to answer the $\mathcal{A}$ queries. The $SimSign$ algorithm is similar to the oracle $Sign$ in the $\mathbf{Exp}^5$.

After receiving the forgery from $\mathcal{A}$, $\mathcal{B}$ verifies this forgery by itself. Assume that $\mathcal{A}$ delivers a valid forgery $(m, \langle r, s\rangle)$ (we denote it as $(\widetilde{m}, \langle\widetilde{r}, \widetilde{s}\rangle)$), i.e. the line 15 is reached. This means that the set Π contains the pair $(\widetilde{R}.x, \widetilde{r}')$: either this pair was already in the Π before verification check in line 13 or it was saved after $SimH_2$ call during this check. There are two possible cases. If $(\widetilde{R}.x, \widetilde{r}') \in \Pi^O$, the forgery is already valid with respect to the oracle H_2^* and $\mathcal{B}$ can simply forward it to its own challenger. If $(\widetilde{R}.x, \widetilde{r}') \in \Pi^S$, $\mathcal{B}$ can recover the signing key d as described below and construct the new forgery with the $Sign^{\mathcal{B}}$ algorithm.

Note that the set Π^S contains only such pairs $(R.x, r')$ that result in the valid signatures $\langle r, s\rangle$. This is provided by the $\mathbf{Exp}^5$ modification of the $Sign$ oracle. Thus, if $(R.x, r') \in \Pi^S$ the adversary $\mathcal{B}$ can search through $\mathcal{L}$ and find element $(m_1, \langle r_1, s_1\rangle)$, which was established during $\mathcal{A}$ signing queries, meanwhile r_1' and R_1 corresponding to $(m_1, \langle r_1, s_1\rangle)$ satisfy: $r_1' = \widetilde{r}'$, $R_1.x = \widetilde{R}.x$. This search can be realized since it's possible to find all elements in $\mathcal{L}$ with $r_1 = \phi(\widetilde{r}')$, compute $e_1 = H_1(m_1)$ and $R_1 = e_1^{-1}s_1P - e_1^{-1}r_1Q$ and check whether $R_1.x = \widetilde{R}.x$.

The $R_1.x = \widetilde{R}.x$ implies $R_1 = \pm\widetilde{R}$ and thus $k_1 = \pm\widetilde{k}$. So the following linear equation system holds:

$$\begin{cases} \widetilde{s} & = \widetilde{k}\widetilde{e} + d\phi(\widetilde{r}'); \\ s_1 & = \pm\widetilde{k}e_1 + d\phi(\widetilde{r}'); \end{cases}$$

for $\widetilde{e} = H_1(\widetilde{m}), e_1 = H_1(m_1)$. There are two unknown variables $\widetilde{k}$ and d in the system above. Moreover, $\phi(\widetilde{r}') \neq 0$ due to the definition of ϕ. This system has a unique solution whenever $\widetilde{e} \neq \pm e_1$. Observe that case $\widetilde{e} = \pm e_1$ and thus $H_1(\widetilde{m}) = \pm H_1(m_1)$ is excluded by lines 5, 6, 7 if $\widetilde{m} \neq m_1$. The $\widetilde{m} = m_1$ condition (together with $\widetilde{r} = r_1$ condition) implies $(\widetilde{m}, \langle\widetilde{r}, \widetilde{s}\rangle) = (m_1, \langle r_1, s_1\rangle)$ and thus is excluded by line 8. Summing all, we can always compute d if the pair $(\widetilde{R}.x, \widetilde{r}')$ belongs to Π^S.

The only remaining challenge is constructing valid forgery by $\mathcal{B}$ using signing key d. $\mathcal{B}$ invokes $Sign^{\mathcal{B}}$ procedure for the message $\widetilde{m}$ that merely repeats the GOST-H.Sig procedure except for $s = 0$ case. In this case $\mathcal{B}$ constructs the forgery for message m_1, found on the previous step, using the same value of k (and r consequently). We claim that s_1 is always nonzero. This follows from the fact that $\widetilde{e} \neq \pm e_1$ as discussed above and thus $s_1 =$

$ke_1 + dr = ke_1 + (s - k\widetilde{e}) = k(e_1 - \widetilde{e}) \neq 0$. Note that [15] does not consider $s = 0$ case.

We conclude that if $\mathcal{A}$ delivers a valid forgery $(\widetilde{m}, \langle \widetilde{r}, \widetilde{s} \rangle)$ to $\mathcal{B}$, $\mathcal{B}$ delivers a valid forgery to its own challenger and

$$\Pr\left[\mathbf{Exp}^5(\mathcal{A}) \Rightarrow 1\right] = \Pr\left[\mathbf{Exp}^{\text{SUF-KO}}_{\mathsf{GOST\text{-}H}}(\mathcal{B}) \Rightarrow 1\right].$$

All in all we proved:

$$\begin{aligned}\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{GOST\text{-}H}}(\mathcal{A}) - \mathsf{Adv}^{\text{SUF-KO}}_{\mathsf{GOST\text{-}H}}(\mathcal{B}) = \Pr\left[\mathbf{Exp}^{\text{SUF-CMA}}_{\mathsf{GOST\text{-}H}}(\mathcal{A}) \Rightarrow 1\right] - \\ - \Pr\left[\mathbf{Exp}^{\text{SUF-KO}}_{\mathsf{GOST\text{-}H}}(\mathcal{B}) \Rightarrow 1\right] = \left(\Pr\left[\mathbf{Exp}^0(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^5(\mathcal{A}) \Rightarrow 1\right]\right) + \\ + \left(\Pr\left[\mathbf{Exp}^5(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^{\text{SUF-KO}}_{\mathsf{GOST\text{-}H}}(\mathcal{B}) \Rightarrow 1\right]\right) \leqslant \\ \leqslant \mathsf{Adv}^{\text{SCR}}_{\mathsf{H}_1}(\mathcal{C}) + \frac{(2Q_O + Q_S + 1)Q_S}{q - 1}.\end{aligned}$$

Note that the number of queries made by $\mathcal{B}$ to the H_2^* oracle is at most $Q_O + 2$.

The adversary $\mathcal{B}$ needs the same amount of computational resources as $\mathcal{C}$, but it also generates new signature in some cases. Thus, $\mathcal{B}$ uses at most $c((Q_S + 3)T^V_{\mathsf{GOST\text{-}H}} + Q_O)$ additional computational resources.

□

B.3 ECDLP **to** SUF-KO **reduction**

Theorem 5. *Let $\mathcal{B}$ be an adversary with time complexity at most T in the* SUF-KO *model for the* $\mathsf{GOST\text{-}H}$ *scheme, making at most Q_O queries to the H_2 oracle. Then there exists an adversary $\mathcal{D}$ that solves the* ECDLP *problem and exists an adversary $\mathcal{M}$ that breaks the signum-relative division resistant property of* H_1*, such that:*

$$\mathsf{Adv}^{\text{SUF-KO}}_{\mathsf{GOST\text{-}H}}(\mathcal{B}) \leqslant \sqrt{Q_O\left(Q_O \cdot \mathsf{Adv}^{\text{SDR}}_{\mathsf{H}_1}(\mathcal{M}) + \mathsf{Adv}^{\text{ECDLP}}_{\mathbb{G}}(\mathcal{D})\right)} + \frac{Q_O + 1}{2^b}.$$

Furthermore, the time complexities of $\mathcal{D}$ and $\mathcal{M}$ are at most $2T + 2c(Q_O + T^V_{\mathsf{GOST\text{-}H}})$, where $T^V_{\mathsf{GOST\text{-}H}}$ is computational resources needed to verify one signature by the $\mathsf{GOST\text{-}H.Vf}$ *procedure, c is a constant that depends only on a model of computation and a method of encoding.*

Proof. Let $\mathbf{Exp}^0$ denote the original security experiment as defined in the SUF-KO security model definition (see Figure 5). We fix $\mathcal{B}$ – the adversary that makes forgery for the $\mathsf{GOST\text{-}H}$ scheme in the SUF-KO model. The adversary has the access to the random oracle H_2, we assume that adversary can make at most Q_O queries to this oracle.

Using the same trick as in Section B.2, we define $\mathbf{Exp}^1$ similar to the $\mathbf{Exp}^0$ but with the H_2 implemented by «lazy sampling» (see Figure 5). As before,

$$\Pr\left[\mathbf{Exp}^0(\mathcal{B}) \Rightarrow 1\right] = \Pr\left[\mathbf{Exp}^1(\mathcal{B}) \Rightarrow 1\right].$$

The $\mathbf{Exp}^2$ is the modification of the $\mathbf{Exp}^1$ in the following way: values β_j, $j = 1, \ldots, Q_O + 1$, are sampled during experiment initializing phase and H_2 oracle just translates them one by one responding to the queries (see Figure 5). Note that additional β_{Q_O+1} value is sampled since challenger queries the H_2 oracle on the finalization step and one more β is used if Π doesn't contain $(R.x, \cdot)$ element (see line 13). We introduce flag flg to indicate $(R.x, \cdot) \notin \Pi$ and abort experiment in case when β_{Q_O+1} matches $\phi^{-1}(r)$ (see line 14). We estimate the difference between $\mathbf{Exp}^1$ and $\mathbf{Exp}^2$ by estimating the probability of this event. Note that this step differs from [15].

$$\Pr\left[\beta_{Q_O+1} \xleftarrow{\mathcal{U}} \{0,1\}^b;\ \beta_{Q_O+1} = \phi^{-1}(r)\right] \leqslant \frac{1}{2^b}.$$

Therefore, we obtain the following bound:

$$\Pr\left[\mathbf{Exp}^1(\mathcal{B}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^2(\mathcal{B}) \Rightarrow 1\right] \leqslant \frac{1}{2^b}.$$

Construction of algorithm C. We construct a deterministic algorithm C that takes a vector $(Q, \beta_1, \ldots, \beta_{Q_O}; \rho)$ as input, invokes the adversary $\mathcal{B}$ on input Q and a random tape derived from ρ and processes the queries to the H_2 oracle as they are processed in the $\mathbf{Exp}^2$ (see Figure 6). Note that random choice in line 4 is made with randomness derived from ρ. Here and after we write «**abort** » as a shortcut for «**return** $\perp$». If $\mathcal{B}$ aborts also C aborts. As $\mathcal{B}$ returns a forgery to C, C validates it and finds the index $j \in \{1, \ldots, Q_O\}$ of the corresponding query to the H_2 oracle. Note that this index always exists cause C aborts if $(R.x, \cdot) \notin \Pi$ before the verification check in line 11. Based on the above,

$$acc = \Pr\left[Q \xleftarrow{\mathcal{U}} \mathbb{G}^*, \beta_1, \ldots, \beta_{Q_O} \xleftarrow{\mathcal{U}} \{0,1\}^b;\ \mathsf{C}(Q, \beta_1, \ldots, \beta_{Q_O}) \not\Rightarrow \perp\right] = \\ = \Pr\left[\mathbf{Exp}^2(\mathcal{B}) \Rightarrow 1\right].$$

Therefore,

$$\Pr\left[\mathbf{Exp}^{\mathsf{SUF\text{-}KO}}_{\mathsf{GOST\text{-}H}}(\mathcal{B}) \Rightarrow 1\right] - acc = \left(\Pr\left[\mathbf{Exp}^0(\mathcal{B}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^1(\mathcal{B}) \Rightarrow 1\right]\right) + \\ + \left(\Pr\left[\mathbf{Exp}^1(\mathcal{B}) \Rightarrow 1\right] - \Pr\left[\mathbf{Exp}^2(\mathcal{B}) \Rightarrow 1\right]\right) + \left(\Pr\left[\mathbf{Exp}^2(\mathcal{B}) \Rightarrow 1\right] - acc\right) \leqslant \frac{1}{2^b}.$$

$\mathbf{Exp}^0(\mathcal{B}) = \mathbf{Exp}^{\text{SUF-KO}}_{\mathsf{GOST\text{-}H}}(\mathcal{B})$

```
1 :  $d \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
2 :  $Q \leftarrow dP$
3 :  $H_2 \xleftarrow{\mathcal{U}} Func(\mathbb{Z}_p, \{0,1\}^b)$
........Setup completed........
4 :  $(m, \langle r, s \rangle) \xleftarrow{\$} \mathcal{B}^{H_2}(Q)$
5 :  if $s = 0$ : abort
6 :  $e \leftarrow H_1(m)$
7 :  if $e = 0 : e \leftarrow 1$
8 :  $R \leftarrow e^{-1}sP - e^{-1}rQ$
9 :  if $\phi(H_2(R.x)) \neq r$ : abort
10 : return 1
```

Oracle $H_2(\alpha)$

```
1 :  return $H_2(\alpha)$
```

$\mathbf{Exp}^1(\mathcal{B})$

```
1 :  $d \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
2 :  $Q \leftarrow dP$
3 :  $\Pi \leftarrow \emptyset$
........Setup completed........
4 :  $(m, \langle r, s \rangle) \xleftarrow{\$} \mathcal{B}^{H_2}(Q)$
5 :  if $s = 0$ : abort
6 :  $e \leftarrow H_1(m)$
7 :  if $e = 0 : e \leftarrow 1$
8 :  $R \leftarrow e^{-1}sP - e^{-1}rQ$
9 :  if $\phi(H_2(R.x)) \neq r$ : abort
10 : return 1
```

Oracle $H_2(\alpha)$

```
1 :  if $(\alpha, \cdot) \in \Pi$ :
2 :      return $\Pi(\alpha)$
3 :  $\beta \xleftarrow{\mathcal{U}} \{0,1\}^b$
4 :  $\Pi \leftarrow \Pi \cup \{(\alpha, \beta)\}$
5 :  return $\beta$
```

$\mathbf{Exp}^2(\mathcal{B})$

```
1 :  $d \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
2 :  $Q \leftarrow dP$
3 :  $\Pi \leftarrow \emptyset$
4 :  $flg \leftarrow \mathsf{false}$
5 :  $i \leftarrow 0$
6 :  $\beta_1, \ldots, \beta_{Q_O+1} \xleftarrow{\mathcal{U}} \{0,1\}^b$
........Setup completed........
7 :  $(m, \langle r, s \rangle) \xleftarrow{\$} \mathcal{B}^{H_2}(Q)$
8 :  if $s = 0$ : abort
9 :  $e \leftarrow H_1(m)$
10 : if $e = 0 : e \leftarrow 1$
11 : $R \leftarrow e^{-1}sP - e^{-1}rQ$
12 : if $(R.x, \cdot) \in \Pi : flg \leftarrow \mathsf{true}$
13 : if $\phi(H_2(R.x)) \neq r$ : abort
14 : if $flg = \mathsf{false}$ : abort
15 : return 1
```

Oracle $H_2(\alpha)$

```
1 :  if $(\alpha, \cdot) \in \Pi$ :
2 :      return $\Pi(\alpha)$
3 :  $i \leftarrow i + 1$
4 :  $\beta \leftarrow \beta_i$
5 :  $\Pi \leftarrow \Pi \cup \{(\alpha, \beta)\}$
6 :  return $\beta$
```

Figure 5: The $\mathbf{Exp}^0$, $\mathbf{Exp}^1$ and $\mathbf{Exp}^2$ for the adversary $\mathcal{B}$ for the GOST-H scheme in the SUF-KO model

$\mathsf{C}(Q, \beta_1, \ldots, \beta_{Q_O}; \rho)$

```
1 :  $\Pi \leftarrow \emptyset$
2 :  $flg \leftarrow \mathsf{false}$
3 :  $i \leftarrow 0$
4 :  $\beta_{Q_O+1} \xleftarrow{\mathcal{U}} \{0,1\}^b$
5 :  $(m, \langle r, s \rangle) \leftarrow \mathcal{B}^{SimH_2}(Q; \rho)$
6 :  $\textbf{if } s = 0 : \textbf{abort}$
7 :  $e \leftarrow H_1(m)$
8 :  $\textbf{if } e = 0 : e \leftarrow 1$
9 :  $R \leftarrow e^{-1}sP - e^{-1}rQ$
10 : $\textbf{if } (R.x, \cdot) \in \Pi : flg \leftarrow \mathsf{true}$
11 : $\textbf{if } \phi(SimH_2(R.x)) \neq r : \textbf{abort}$
12 : $\textbf{if } flg = \mathsf{false} : \textbf{abort}$
13 : $r' \leftarrow \phi^{-1}(r)$
14 : $\text{find } \ j \in \{1, \ldots, Q_O\} : r' = \beta_j$
15 : $\textbf{return } \ (j, m, \langle r, s \rangle)$
```

$\mathsf{Fork_C}(Q)$

```
1 :  Pick random coins $\rho$ for $\mathsf{C}$
2 :  $\beta_1, \ldots, \beta_{Q_O} \xleftarrow{\mathcal{U}} \{0,1\}^b$
3 :  $(j, m_1, \langle r_1, s_1 \rangle) \leftarrow \mathsf{C}(Q, \beta_1, \ldots, \beta_{Q_O}; \rho)$
4 :  $\beta'_j, \ldots, \beta'_{Q_O} \xleftarrow{\mathcal{U}} \{0,1\}^b$
5 :  $\textbf{if } \beta_j = \beta'_j : \textbf{abort}$
6 :  $\big(j', m_2, \langle r_2, s_2 \rangle\big) \leftarrow \mathsf{C}(Q, \beta_1, \ldots, \beta_{j-1}, \beta'_j, \ldots, \beta'_{Q_O}; \rho)$
7 :  $\textbf{if } j \neq j' : \textbf{abort}$
8 :  $\textbf{return } \ (m_1, \langle r_1, s_1 \rangle, m_2, \langle r_2, s_2 \rangle)$
```

$\mathcal{D}(Q)$

```
1 :  $(m_1, \langle r_1, s_1 \rangle, m_2, \langle r_2, s_2 \rangle) \leftarrow \mathsf{Fork_C}(Q)$
2 :  $\textbf{if } H_1(m_1)/r_1 = \pm H_1(m_1)/r_2 : \textbf{abort}$
3 :  compute $d$
4 :  $\textbf{return } d$
```

Figure 6: The C algorithm that uses the adversary $\mathcal{B}$ for the $\mathsf{GOST\text{-}H}$ scheme in the SUF-KO model; the $\mathsf{Fork_C}$ algorithm that uses C algorithm and the adversary $\mathcal{D}$ that solves ECDLP problem using $\mathsf{Fork_C}$ algorithm

The algorithm C invokes the adversary $\mathcal{B}$, processes at most Q_O queries to H_2 oracle and verifies the forgery obtained from $\mathcal{B}$. Thus, computational complexity of C is at most $T + c(Q_O + T^V_{\mathsf{GOST\text{-}H}})$, where T is the computational resources of $\mathcal{B}$, $T^V_{\mathsf{GOST\text{-}H}}$ is computational resources needed to verify one signature by the $\mathsf{GOST\text{-}H.Vf}$ procedure, c is a constant that depends only on a model of computation and a method of encoding.

We apply the forking lemma (see [15]) and construct the forking algorithm Fork_C (see Figure 6). If C aborts also Fork_C aborts. According to the forking lemma the probability frk that Fork_C terminates without aborting can be estimated as

$$\Pr\left[Q \overset{\mathcal{U}}{\leftarrow} \mathbb{G}^*;\ \mathsf{Fork}_\mathsf{C}(Q) \not\Rightarrow \perp\right] = frk \geq acc\left(\frac{acc}{Q_O} - \frac{1}{2^b}\right).$$

Construction of adversary $\mathcal{D}$. Finally we construct the adversary $\mathcal{D}$ that solves the ECDLP problem (see Figure 6). At first $\mathcal{D}$ invokes Fork_C algorithm with the same input as its own input. If Fork_C aborts also $\mathcal{D}$ aborts. Obtaining two pairs (message, signature) from Fork_C, $\mathcal{D}$ checks whether the condition in line 2 holds and otherwise computes d with the algorithm described below.

Using the pairs obtained from Fork_C adversary $\mathcal{D}$ computes $e_1 = H_1(m_1), e_2 = H_1(m_2)$ and constructs the following linear system of equations:

$$\begin{cases} R_1 & = e_1^{-1}s_1P - e_1^{-1}r_1Q; \\ R_2 & = e_2^{-1}s_2P - e_2^{-1}r_2Q; \\ r_1 & \neq r_2. \end{cases}$$

By construction of Fork_C second execution of C differs only since the j-th query of $\mathcal{B}$ to the H_2 oracle. Therefore the j-th input $\alpha = R.x$ to the H_2 oracle was the same in two executions and we claim that $R_1.x = R_2.x$ and thus $R_1 = \pm R_2$. We transform the system above to the following equation

$$e_1^{-1}s_1P - e_1^{-1}r_1Q = \pm e_2^{-1}s_2P \mp e_2^{-1}r_2Q;$$

and compute d by the following formula:

$$d = \frac{e_1^{-1}s_1 \mp e_2^{-1}s_2}{e_1^{-1}r_1 \mp e_2^{-1}r_2}.$$

Note that condition $e_1^{-1}r_1 \mp e_2^{-1}r_2 \neq 0$ holds due to abort in line 2. Summing all, $\mathcal{D}$ computes d as soon as Fork_C does not abort and condition in line 2 is not met.

Construction of adversary $\mathcal{M}$. We can estimate the probability of aborting in line 2 by constructing an adversary $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2)$ for the signum-relative division resistance property (see Figure 7).

The construction of adversary $\mathcal{M}_1$ is quite similar to lines 1-3 of Fork_C algorithm up to the following difference: adversary $\mathcal{M}_1$ guesses $j^* \in \{1, \ldots, Q_O\}$ (see line 3 of $\mathcal{M}_1$) and puts needed value β to the j^*-position in the C input. If C aborts, also $\mathcal{M}_1$ aborts. Obtaining $(j_1, m_1, \langle r_1, s_1\rangle)$, adversary $\mathcal{M}_1$ checks whether j^* is guessed correctly, and, if so, returns its internal state Γ and m_1 to its own challenger. Adversary $\mathcal{M}_2$ is invoked on input (Γ, β') and simulates lines 4-7 of Fork_C algorithm up to the following difference: it puts β' to the j^*-position in the C input. If C aborts, also $\mathcal{M}_2$ aborts. Once $\mathcal{M}_2$ gets $(j_2, m_2, \langle r_2, s_2\rangle)$, it checks whether $j_1 = j_2$ and, if so, returns m_2 to its own challenger. Adversary $\mathcal{M}$ wins if $H_1(m_1)/\phi(\beta) = \pm H_1(m_2)/\phi(\beta')$.

```
$\mathcal{M}_1(\beta)$
1 :  $d \overset{\mathcal{U}}{\leftarrow} \mathbb{Z}_q^*$
2 :  $Q \leftarrow dP$
3 :  $j^* \leftarrow \{1, \ldots, Q_O\}$
4 :  Pick random coins $\rho$ for $\mathsf{C}$
5 :  $\beta_1, \ldots, \beta_{Q_O} \overset{\mathcal{U}}{\leftarrow} \{0,1\}^b$
6 :  $\beta_{j^*} \leftarrow \beta$
7 :  $(j_1, m_1, \langle r_1, s_1\rangle) \leftarrow \mathsf{C}(Q, \beta_1, \ldots, \beta_{Q_O}; \rho)$
8 :  **if** $j_1 \neq j^*$ : **abort**
9 :  $\Gamma \leftarrow (Q, \beta_1, \ldots, \beta_{j^*}, \rho)$
10 : **return** $(m_1, \Gamma)$
```

```
$\mathcal{M}_2(\Gamma, \beta')$
1 :  $(Q, \beta_1, \ldots, \beta_{j^*}, \rho) \leftarrow \Gamma$
2 :  $\beta'_{j^*}, \ldots, \beta'_{Q_O} \overset{\mathcal{U}}{\leftarrow} \{0,1\}^b$
3 :  $\beta'_{j^*} \leftarrow \beta'$
4 :  **if** $\beta'_{j^*} = \beta_{j^*}$ : **abort**
5 :  $(j_2, m_2, \langle r_2, s_2\rangle) \leftarrow \mathsf{C}(Q, \beta_1, \ldots, \beta_{j^*-1}, \beta'_{j^*}, \ldots, \beta'_{Q_O}; \rho)$
6 :  **if** $j_1 \neq j_2$ : **abort**
7 :  **return** $m_2$
```

Figure 7: The adversary $\mathcal{M}$ for the SDR property that uses algorithm C

Let denote $\mathsf{Adv}_{\mathsf{H}_1}^{\mathrm{SDR}}(\mathcal{M})$ as $\epsilon_{\mathsf{H}_1}^{\mathrm{SDR}}$. Note that the adversary $\mathcal{M}$ wins if it guesses j^* correctly and the abort condition (see line 2 in $\mathcal{D}$'s pseudocode) is met. Then we can estimate the probability of abort condition as

$$\Pr[H_1(m_1)/r_1 = \pm H_1(m_1)/r_2] \leqslant Q_O \epsilon_{\mathsf{H}_1}^{\mathrm{SDR}}.$$

Therefore we obtain the following bound:

$$\delta = \Pr[\mathcal{D} \text{ solves ECDLP}] = \Pr\left[(\mathsf{Fork}_{\mathsf{C}}(Q) \not\Rightarrow \bot) \wedge \left(\frac{H_1(m_1)}{r_1} \neq \pm\frac{H_1(m_2)}{r_2}\right)\right] =$$
$$= \Pr[\mathsf{Fork}_{\mathsf{C}}(Q) \not\Rightarrow \bot] - \Pr\left[(\mathsf{Fork}_{\mathsf{C}}(Q) \not\Rightarrow \bot) \wedge \left(\frac{H_1(m_1)}{r_1} = \pm\frac{H_1(m_2)}{r_2}\right)\right] \geq$$
$$\geq frk - Q_O\epsilon^{\mathrm{SDR}}_{\mathsf{H}_1} \geq acc\left(\frac{acc}{Q_O} - \frac{1}{2^b}\right) - Q_O\epsilon^{\mathrm{SDR}}_{\mathsf{H}_1} = \frac{1}{Q_O}acc^2 - \frac{1}{2^b}acc - Q_O\epsilon^{\mathrm{SDR}}_{\mathsf{H}_1}.$$

By decision of the following inequation:

$$\frac{1}{Q_O}acc^2 - \frac{1}{2^b}acc - (Q_O\epsilon^{\mathrm{SDR}}_{\mathsf{H}_1} + \delta) \leqslant 0.$$

we can bound the *acc* value as:

$$acc \leqslant \frac{Q_O}{2^b} + \sqrt{Q_O\left(Q_O\epsilon^{\mathrm{SDR}}_{\mathsf{H}_1} + \delta\right)}.$$

Summarizing all the results, we obtain the final bound to complete the proof:

$$\mathsf{Adv}^{\text{SUF-KO}}_{\mathsf{GOST\text{-}H}}(\mathcal{B}) = \Pr\left[\mathbf{Exp}^{\text{SUF-KO}}_{\mathsf{GOST\text{-}H}}(\mathcal{B}) \Rightarrow 1\right] \leqslant acc + \frac{1}{2^b} \leqslant$$
$$\leqslant \sqrt{Q_O\left(Q_O \cdot \mathsf{Adv}^{\mathrm{SDR}}_{\mathsf{H}_1}(\mathcal{M}) + \mathsf{Adv}^{\mathrm{ECDLP}}_{\mathbb{G}}(\mathcal{D})\right)} + \frac{Q_O+1}{2^b}.$$

Both $\mathcal{D}$ and $\mathcal{M}$ invoke the algorithm C twice, thus their computational complexities are at most $2T + 2c(Q_O + T^V_{\mathsf{GOST\text{-}H}})$.

□

B.4 Signum-relative division resistance property

In this section we consider the signum-relative division resistance property of H_1 and show that this notion is implied by the standard assumptions: zero resistance and signum-relative preimage resistance properties of H_1.

Let us formally introduce these two properties.

Definition 6 (Zero-resistance property). *For the family of hash functions* H_1

$$\mathsf{Adv}^{\mathrm{ZR}}_{\mathsf{H}_1}(\mathcal{A}) = \Pr\left[m \xleftarrow{\$} \mathcal{A} : H_1(m) = 0\right]$$

Definition 7 (Signum-relative preimage resistance property). *For the family of hash functions* H_1

$$\mathsf{Adv}^{\mathrm{SPR}}_{\mathsf{H}_1}(\mathcal{A}) = \Pr\left[y \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*; m \xleftarrow{\$} \mathcal{A}(y) : H_1(m) = \pm y\right]$$

We construct the adversary $\mathcal{S}$ that breaks the signum-relative preimage property and uses the adversary $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2)$ that breaks the signum-relative division resistance property as a black box (see Figure 8).

```
S(y)
1 : $\beta_1 \xleftarrow{\mathcal{U}} \{0,1\}^b$
2 : $(m_1, \Gamma) \xleftarrow{\$} \mathcal{M}_1(\beta_1)$
3 : **if** $H_1(m_1) = 0$ : **abort**
4 : $\beta_2 \leftarrow \phi^{-1}(y \cdot \phi(\beta_1) \cdot (H_1(m_1))^{-1})$
5 : $m_2 \xleftarrow{\$} \mathcal{M}_2(\Gamma, \beta_2)$
6 : **return** $m_2$
```

Figure 8: The adversary $\mathcal{S}$ for the SPR property that uses the adversary $\mathcal{M}$ for the SDR property

Let denote $H_1(m_1) = 0$ condition as *Event*. We can estimate the probability of *Event* by constructing an adversary $\mathcal{P}$ that breaks the zero-resistance property. Adversary $\mathcal{P}$ simply simulates lines 1-2 of $\mathcal{S}$ pseudocode and wins if the *Event* takes place. Thus,

$$\Pr[\mathit{Event}] = \mathsf{Adv}^{\mathrm{ZR}}_{\mathsf{H}_1}(\mathcal{P}).$$

Consider the distribution on β_2 values. Let denote $\big(y \cdot \phi(\beta_1) \cdot (H_1(m_1))^{-1}\big)$ as γ. As y is chosen randomly from $\mathbb{Z}_q^*$, we claim that γ is distributed uniformly on $\mathbb{Z}_q^*$. Note that $\mathbb{Z}_q^*$ contains less elements than $\mathbb{Z}_{2^{\lceil \log q \rceil}}$ and thus for different values of β_2 the probability $\Pr\left[\gamma \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*;\ \phi^{-1}(\gamma) = \beta_2\right]$ may not be the same. However, we claim that for different values of β_2 this probability will not differ more than by $1/(q-1)$. We find this difference negligible and consider the distribution on β_2 as close to uniform.

If abort condition in line 3 is not met and β_2 is distributed uniformly on $\{0,1\}^b$, the adversary $\mathcal{S}$ realizes the same experiment for $\mathcal{M}$ as in Definition 4. Thus, we can estimate the probability of $\mathcal{S}$ success as

$$\begin{aligned}\Pr[\mathcal{S} \text{ breaks SPR-property}] &= \Pr\left[\overline{\mathit{Event}} \wedge (\mathcal{M} \text{ breaks SDR-property})\right] = \\ = \Pr[\mathcal{M} \text{ breaks SDR-property}] &- \Pr[\mathit{Event} \wedge (\mathcal{M} \text{ breaks SDR-property})] \geqslant \\ &\geqslant \Pr[\mathcal{M} \text{ breaks SDR-property}] - \Pr[\mathit{Event}].\end{aligned}$$

Consequently,

$$\mathsf{Adv}^{\mathrm{SDR}}_{\mathsf{H}_1}(\mathcal{M}) \leqslant \mathsf{Adv}^{\mathrm{SPR}}_{\mathsf{H}_1}(\mathcal{S}) + \mathsf{Adv}^{\mathrm{ZR}}_{\mathsf{H}_1}(\mathcal{P}).$$

C $\mathsf{GenEG_S}$ security

Let $\mathcal{A}$ be an adversary for the $\mathsf{GenEG_S}$ scheme in the SUF-CMA model. We construct the adversary $\mathcal{B}$ for the GenEG scheme in the SUF-CMA model that uses $\mathcal{A}$ as the black box (see Figure 9). Note that $\mathcal{B}$ has the access to its own signing oracle $Sign^*$.

```
B^{Sign*}(Q)
1 : $\mathcal{L} \leftarrow \emptyset$
2 : $(m, \overline{r}^* \| \overline{s}) \xleftarrow{\$} \mathcal{A}^{SimSign}(Q)$
3 : if $(m, \overline{r}^* \| \overline{s}) \in \mathcal{L}$ : abort
4 : $\overline{r} \leftarrow \overline{r}^* \| const$
5 : return $(m, \overline{r} \| \overline{s})$
```

```
SimSign(m)
1 : $cnt \leftarrow 0$
2 : if $cnt > thr$ : return $\perp$
3 : $cnt \leftarrow cnt + 1$
4 : $\overline{r} \| \overline{s} \leftarrow Sign^*(m)$
5 : if $\overline{r} \| \overline{s} = \perp$ : goto 2
6 : if $\mathsf{lsb}_l(\overline{r}) \neq const$ : goto 2
7 : $\overline{r}^* = \mathsf{msb}_{|\overline{r}|-l}(\overline{r})$
8 : $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, \overline{r}^* \| \overline{s})\}$
9 : return $\overline{r}^* \| \overline{s}$
```

Figure 9: The adversary $\mathcal{B}$ for the GenEG scheme in the SUF-CMA model that uses the adversary $\mathcal{A}$ for the $\mathsf{GenEG_S}$ scheme in the SUF-CMA model

Adversary $\mathcal{B}$ invokes $\mathcal{A}$ as a subroutine. $\mathcal{B}$ simulates the $Sign$ oracle for $\mathcal{A}$ with $SimSign$ procedure. Similarly to the $\mathsf{GenEG_S.Sig}$ procedure, $\mathcal{B}$ generates «full» signatures with its own oracle until the $\overline{r}$ component matches the constant vector and truncates $\overline{r}$ before outputting the signature.

Obtaining the forgery from $\mathcal{A}$, $\mathcal{B}$ recovers $\overline{r}$ component by concatenation it with constant vector and forwards it to its own challenger.

If $\mathcal{A}$ makes a valid forgery, $\mathcal{B}$ also makes it. Thus,

$$\Pr\left[\mathbf{Exp}^{\text{SUF-CMA}}_{\mathsf{GenEG_S}}(\mathcal{A}) \Rightarrow 1\right] = \Pr\left[\mathbf{Exp}^{\text{SUF-CMA}}_{\mathsf{GenEG}}(\mathcal{B}) \Rightarrow 1\right].$$

Assume that $\mathcal{A}$ makes at most Q_S queries to the signing oracle. Then $\mathcal{B}$ by construction makes at most $Q_S \cdot thr$ queries to its own signing oracle.

D $\mathsf{GenEG^V}$ security

Let $\mathcal{A}$ be an adversary for the $\mathsf{GenEG^V}$ scheme in the SUF-CMA model. We construct the adversary $\mathcal{B}$ for the GenEG scheme in the SUF-CMA model that uses $\mathcal{A}$ as the black box (see Figure 10). Note that $\mathcal{B}$ has the access to its own signing oracle $Sign^*$.

```
$\mathcal{B}^{Sign^*}(Q)$
1 : $\mathcal{L} \leftarrow \emptyset$
2 : $(m, \overline{r} \| \overline{s}^*) \xleftarrow{\$} \mathcal{A}^{SimSign}(Q)$
3 : **if** $(m, \overline{r} \| \overline{s}^*) \in \mathcal{L}$ : **abort**
4 : $i \leftarrow 0$
5 : **if** $i \geq 2^t$ : **abort**
6 : $\overline{s} \leftarrow \overline{s}^* \| \mathsf{str}_t(i)$
7 : $i \leftarrow i + 1$
8 : $res \leftarrow \mathsf{GenEG.Vf}(Q, m, \overline{r} \| \overline{s})$
9 : **if** $res = 0$ : **goto** 5
10 : **return** $(m, \overline{r} \| \overline{s})$
```

```
$SimSign(m)$
1 : $\overline{r} \| \overline{s} \leftarrow Sign^*(m)$
2 : **if** $\overline{r} \| \overline{s} = \bot$ : **return** $\bot$
3 : $\overline{s}^* \leftarrow \mathsf{msb}_{|\overline{s}|-t}(\overline{s})$
4 : $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, \overline{r} \| \overline{s}^*)\}$
5 : **return** $\overline{r} \| \overline{s}^*$
```

Figure 10: The adversary $\mathcal{B}$ for the GenEG scheme in the SUF-CMA model that uses the adversary $\mathcal{A}$ for the $\mathsf{GenEG}^{\mathsf{V}}$ scheme in the SUF-CMA model

Adversary $\mathcal{B}$ invokes $\mathcal{A}$ as a subroutine. $\mathcal{B}$ simulates the $Sign$ oracle for $\mathcal{A}$ with $SimSign$ procedure. Similarly to the $\mathsf{GenEG}^{\mathsf{V}}.\mathsf{Sig}$ procedure, $\mathcal{B}$ truncates $\overline{s}$ component before outputting the signature.

Obtaining the forgery from $\mathcal{A}$, adversary $\mathcal{B}$ iterates through all possible variants of $\overline{s}$ and verifies signature until the verification procedure stops with 1. Note that $\mathcal{B}$ needs at most $2^t \cdot T^V_{\mathsf{GenEG}}$ additional computational resources to recover the $\overline{s}$ component, where T^V_{GenEG} is computational resources needed to verify one signature by the GenEG.Vf procedure.

If $\mathcal{A}$ makes a valid forgery, $\mathcal{B}$ also makes it. Thus,

$$\Pr\left[\mathbf{Exp}^{\text{SUF-CMA}}_{\mathsf{GenEG}^{\mathsf{V}}}(\mathcal{A}) \Rightarrow 1\right] = \Pr\left[\mathbf{Exp}^{\text{SUF-CMA}}_{\mathsf{GenEG}}(\mathcal{B}) \Rightarrow 1\right].$$

Adversary $\mathcal{B}$ by construction makes the same as $\mathcal{A}$ number of queries to its own signing oracle.