Two-party GOST in two parts: fruitless search and fruitful synthesis

Liliya Akhmetzyanova, Evgeny Alekseev, Alexandra Babueva,
Lidiia Nikiforova and Stanislav Smyshlyaev

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

Abstract

In the current paper we investigate the possibility of designing secure two-party signature scheme with the same verification algorithm as in the Russian standardized scheme (GOST scheme). We solve this problem in two parts. The first part is a (fruitless) search for an appropriate scheme in the literature. It turned out that all existing schemes are insecure in the strong security models. The second part is a synthesis of new signature scheme and ends fruitfully. We synthesize a new two-party GOST signature scheme, additionally using the commitment scheme, guided by the features of the GOST signature scheme, as well as the known attacks on existing schemes. We prove that this scheme is secure in a bijective random oracle model in the case when one of the parties is malicious under the assumption that the classical GOST scheme is unforgeable in a bijective random oracle model and the commitment scheme is modelled as a random oracle.

Keywords: two-party signature, GOST signature.

1 Introduction

Electronic document management systems become a common daily occurrence in the modern world. Signature scheme is a fundamental component of these systems. The systems involve the client, who owns a private signing key, and the server, who manages the documents. The server sends the document to the client, who checks it and signs. It is highly desirable to implement the client side at the user mobile device to make the information system as user-friendly as possible. There is a problem of secure storage of the private key on a mobile device, since it is easy to gain physical access to the device, for example, as a result of theft. If the adversary gets access to the private key, it will be able to sign documents on behalf of the user. So, we need a way to protect the private key stored on the mobile device.

One method of protection is to use the so-called two-party signature scheme instead of the classical signature scheme. This method involves the private key sharing between the client and the server and generating the signature as a result of an interactive protocol run between them. We assume that no trusted party is involved in this process. Such protocol should not allow either party to create a signature without interacting with the

other party. In particular, the server can not sign any document without the owner of the signing key. At the same time in case of theft of the user's device, the adversary gets access to only one part of the key and needs to interact with the server to create a signature. Note that the server can notify the user about each execution of the protocol via an outside channel, for example, by e-mail. The user whose mobile device has been stolen can report this to the server and forbid the possibility of creating a signature.

This method of protection should remain completely transparent to all external systems that can potentially use the generated signature. That is, it should not differ from the classic signature generated when the key is fully stored on the user's device. This means that the verification algorithm should be the same as in the classical scheme. We use the Russian signature scheme defined in [8, 9, 10, 11] (hereinafter — GOST scheme) as a classical signature scheme. Thus, to implement the described method of protecting the private key, we need a two-party signature scheme with the same verification algorithm as in the GOST scheme.

In literature there are a number of schemes [4, 16, 13, 14, 1, 20] based on GOST signature equation, in which the signature is generated by several signers. It is not only two-party schemes, but also schemes for more participants: collective signature schemes (for n parties) and threshold signature schemes where any subset of at least t out of the n parties can produce a valid signature. Note that such schemes are the extensions of two-party schemes, therefore they can also be used for private key protection.

However, it turned out that all existing schemes are not suitable for solving our problem. Some of them are proven secure in the weak security models while others are vulnerable to the attacks. Let's consider these schemes. The threshold signature scheme proposed in [1] uses a third trusted party to form the signature. The signature scheme proposed in [13] uses a new secret sharing algorithm for key generation and signing protocols and is proven secure only against passive adversary. The scheme proposed in [20] appeared to be insecure if the adversary is given the opportunity to open parallel sessions of the signing protocol. In this paper, we build a ROS-style attack [2] on this scheme (see Appendix A.1). In [4, 16, 14] there is no description of the distributed key generation protocol for the proposed scheme. It is not clear how to implement it if no third trusted party is involved. Moreover, the signing protocol of this scheme is vulnerable to a ROS attack.

We synthesize a new two-party GOST signature scheme, additionally using the commitment scheme, guided by the features of the GOST signature scheme, as well as the known attacks on existing schemes. Our scheme does not use any non-standard cryptographic mechanisms such as homomorphic encryption. Section 3 presents the design rationale of this scheme. A formal description of the scheme is provided in the Section 4. We prove that this scheme is secure in a bijective random oracle model in the case when one of the parties is malicious under the assumption that the classical GOST scheme is unforgeable in a bijective random oracle model and the commitment scheme is modelled as a random oracle. Our proof is based on the security proof of the GOST signature scheme presented in [5] and the security proof for the two-party Schnorr signature scheme [17]. A description of the security model and the main result are presented in the Section 5.

2 Basic notations and definitions

By $\{0,1\}^*$ we denote the set of all bit strings of finite length including the empty string. If p is a prime number then the set $\mathbb{Z}_p$ is a finite field of size p. We assume the canonic representation of the elements in $\mathbb{Z}_p$ as integers in the interval $[0 \dots p-1]$. 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 x-coordinate of the point $R \in \mathbb{G}$ as $R.x$. We denote by H the hash function that maps binary strings of arbitrary length to the binary string of length h.

If the value s is chosen from a set S uniformly at random, then we denote $s \xleftarrow{\mathcal{U}} S$. 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 an algorithm A we denote $x \leftarrow A$.

The signature scheme SS is determined by three algorithms:

- $(d, Q) \leftarrow \mathsf{KGen}()$: a probabilistic key generation algorithm that returns the signature key pair (d, Q), where d is a private signing key, Q is a public verifying key.
- $\sigma \leftarrow \mathsf{Sign}(d, m)$: a probabilistic signing algorithm that takes a signing key d and a message m as an input and outputs a signature σ for the message m.
- $b \leftarrow \mathsf{Verify}(Q, m, \sigma)$: a (deterministic) verification algorithm that takes a public verifying key Q, a message m and a signature σ as an input and outputs 1 if σ is valid and 0 otherwise.

The two-party signature scheme $\mathsf{2p\text{-}SS}$ is determined by three algorithms:

- $((d_1, Q), (d_2, Q)) \leftarrow \mathsf{KGen}\langle \mathbf{P_1}(), \mathbf{P_2}() \rangle$: an interactive key generation protocol that is run between a party $\mathbf{P_1}$ and a party $\mathbf{P_2}$; for $i \in \{1, 2\}$ $\mathbf{P_i}$ outputs it's private key d_i and a public verifying key Q.
- $(\sigma, \sigma) \leftarrow \mathsf{Sign}\langle \mathbf{P_1}(d_1, Q, m), \mathbf{P_2}(d_2, Q, m) \rangle$: an interactive signing protocol that is run between a party $\mathbf{P_1}$ and a party $\mathbf{P_2}$; for $i \in \{1, 2\}$ $\mathbf{P_i}$ takes it's private key d_i, a public verifying key Q and a message m as an input and outputs a signature σ for the message m if the interaction completes successfully and $\perp$ otherwise.
- $b \leftarrow \mathsf{Verify}(Q, m, \sigma)$: a (deterministic) verification algorithm that takes a public verifying key Q, a message m and a signature σ as an input and outputs 1 if σ is valid and 0 otherwise.

The commitment scheme is determined by two algorithms:

- $(op, comm) \leftarrow \mathsf{Cmt}(m)$: a commitment algorithm that takes message $m \in \{0,1\}^*$ as an input and outputs a commitment $comm \in \{0,1\}^n$ and an opening value $op \in \{0,1\}^\kappa$.
- $b \leftarrow \mathsf{Open}(comm, m, op)$: a (deterministic) opening algorithm that takes a commitment $comm \in \{0,1\}^n$, a message $m \in \{0,1\}^*$ and an opening value $op \in \{0,1\}^\kappa$ and outputs 1 if $(op, comm)$ is valid on m and 0 otherwise.

3 Design rationale

The GOST signature is a pair (r, s):

$$s = ke + dr,\ r = R.x \bmod q = (kP).x \bmod q,$$

where k is selected uniformly from $\mathbb{Z}_q^*$, $d \in \mathbb{Z}_q^*$ is a private key, e is the hash of the message m. The secret parameters are the long-term signing key d and the ephemeral value k.

Two-party signature scheme implies that both parties contribute to the generation of all secret parameters. Note that the signature equation is linear with respect to secret parameters d and k. Thus, the straightforward way is to use additive secret sharing of these parameters: $k = k_1 + k_2$, $d = d_1 + d_2$. Then the signature (r, s) is formed as:

$$s = (k_1 + k_2)e + (d_1 + d_2)r = (k_1 e + d_1 r) + (k_2 e + d_2 r),$$

$$r = (R_1 + R_2).x \mod q = (k_1 P + k_2 P).x \mod q.$$

Consider the naive version of the two-party GOST signature scheme between participants $\mathbf{P_1}$ and $\mathbf{P_2}$. The key generation protocol KGen and the signing protocol Sign are shown at Figure 1 and Figure 2 respectively.

Key generation protocol. At first, let's consider the KGen protocol. We claim that this key generation algorithm is not secure. Indeed, the party $\mathbf{P_2}$ can choose Q_2 depending on Q_1 in such a way that it will know the discrete logarithm of the final public key Q, i.e. the private key d. For example, the party $\mathbf{P_2}$ can set $Q_2 = P - Q_1$. Then, $Q = Q_1 + Q_2 = P$, $d = 1$.

One way to protect against this attack is to use a commitment scheme just like in the two-party Schnorr signature scheme [17]. Instead of sending Q_1, the party $\mathbf{P_1}$ can send the commitment to the value Q_1. Then party $\mathbf{P_2}$ does not know any information about Q_1 due to the «hiding» property of the commitment scheme and generates Q_2 independently of Q_1. The party $\mathbf{P_1}$ cannot change Q_1 after it receives Q_2 from the party $\mathbf{P_2}$ due to the «binding» property of the commitment scheme.

Another way of protection is to use the multiplicative method of the private key d sharing as in the scheme proposed in [20]. The simple version of the algorithm is shown at Figure 3. The party $\mathbf{P_2}$ can also set the Q_2 value depending on Q_1. However in such case it only knows how d depends on d_1, but does not know the d value itself because of unpredictability of d_1. For example, the party $\mathbf{P_2}$ can set $Q_2 = Q_1$. Then, the party $\mathbf{P_1}$ calculates $Q = d_1 \cdot Q_1 = d_1^2 \cdot P$.

Note that in case of the multiplicative key sharing there is no obvious way how to further use the key shares to create a signature. Specifically, participants must calculate the value $d_1 \cdot d_2 \cdot r$ without revealing the secret to the other party. The authors of the paper [20] use the additively homomorphic encryption scheme based on factoring problem for such calculation.

Since we strive not to use non-standard cryptographic mechanisms, we decide to use the additive secret sharing with the commitment scheme.

KGen

$\mathbf{P_1}$ ( )		$\mathbf{P_2}$ ( )
$d_1 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$		
$Q_1 \leftarrow d_1 \cdot P$		
	$\xrightarrow{Q_1}$	
		$d_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
		$Q_2 \leftarrow d_2 \cdot P$
		$Q \leftarrow Q_1 + Q_2$
	$\xleftarrow{Q_2}$	
$Q \leftarrow Q_1 + Q_2$		
return (d_1, Q)		**return** (d_2, Q)

Figure 1: The key generation protocol of the naive version of the two-party GOST

Sign

$\mathbf{P_1}$ (d_1, Q, m)		$\mathbf{P_2}$ (d_2, Q, m)
$e \leftarrow \mathrm{H}(m)$		$e \leftarrow \mathrm{H}(m)$
$k_1 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$		
$R_1 \leftarrow k_1 \cdot P$		
	$\xrightarrow{R_1}$	
		$k_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
		$R_2 \leftarrow k_2 \cdot P$
		$r \leftarrow (R_1 + R_2).x \mod q$
		$s_2 \leftarrow k_2 \cdot e + d_2 \cdot r$
	$\xleftarrow{R_2, s_2}$	
$r \leftarrow (R_1 + R_2).x \mod q$		
$s_1 \leftarrow k_1 \cdot e + d_1 \cdot r$		
$s \leftarrow s_1 + s_2$		
	$\xrightarrow{s_1}$	
		$s \leftarrow s_1 + s_2$
return $\langle r, s \rangle$		**return** $\langle r, s \rangle$

Figure 2: The signing protocol of the naive version of the two-party GOST

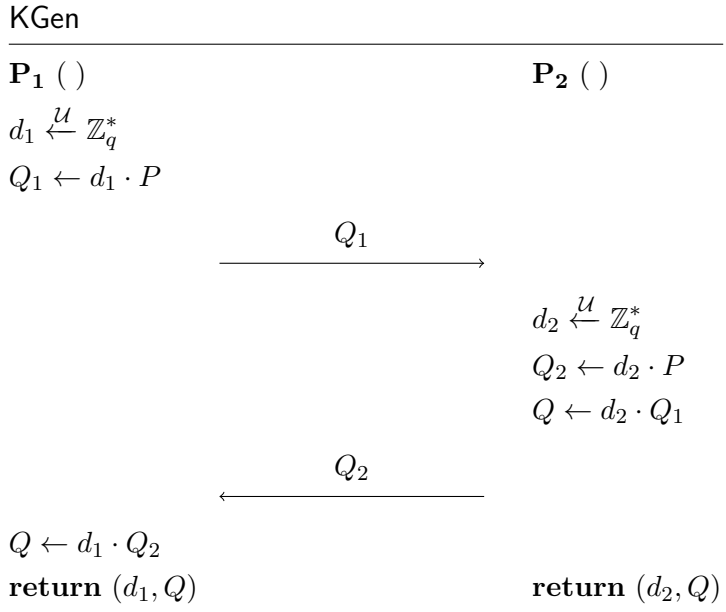

Figure 3: The KGen protocol of the scheme from [20]

Signing protocol. Let's consider the Sign protocol from Figure 2 which is the same as in the schemes proposed in [4, 16, 14]. We claim that it is not secure, since it is vulnerable to the ROS-style attack. Note that in the scheme proposed in [4] party $\mathbf{P_2}$ sends r instead of R_2, but these are equivalent cases, since the R_2 value can be restored from r and R_1.

The original ROS attack was proposed in [2], the authors show that it is applicable to some threshold signature schemes [7, 12]. The attack works if the one party is given the opportunity to open $l \geqslant \lceil \log q \rceil$ parallel sessions of signing protocol with the other party. Let's discuss the features of signing protocol from Figure 2 that make the attack applicable. The main observation is that one party can select its parameters when it knows the parameters selected by the other party. Indeed, the party $\mathbf{P_2}$ can open l parallel sessions with $\mathbf{P_1}$, receive l points $R_1^1, \ldots, R_1^l$ and construct the corresponding R_2 points in some specific way dependent on R_1 values. Note that in the scheme from [20] the party $\mathbf{P_2}$ also can vary R_2 after receiving R_1 from the party $\mathbf{P_1}$. We provide an explicit description of ROS-style attack on this scheme in Appendix A.1 and some modification of this attack for scheme defined at Figure 2 in Appendix A.2.

We use the commitment scheme to protect against this attack. Instead of sending R_1, the party $\mathbf{P_1}$ can send the commitment to the value R_1. Then party $\mathbf{P_2}$ does not know any information about R_1 due to the «hiding» property of the commitment scheme and generates R_2 independently of R_1. The party $\mathbf{P_1}$ cannot change R_1 after it receives R_2 from the party $\mathbf{P_2}$ due to the «binding» property of the commitment scheme. Consequently, each party cannot vary any parameters after receiving the parameters selected by the other party.

Note that up to this point we have assumed that the message initially exists on both sides, i.e. given them as an input. In practice, one of the parties usually forwards the message to the other. It is important that each party captures the message before it learns the parameters of the other party to protect against the ROS-style attack. We provide the description of the attack on the modification of discussed signing protocol in which the party $\mathbf{P_1}$ selects message m and sends it to the party $\mathbf{P_2}$ after receiving R_2 in Appendix A.3.

4 Two-party GOST

In this section we describe the two-party signature scheme 2p-GOST. It is based on the GOST signature scheme.

The key generation protocol KGen and the signing protocol Sign use a commitment scheme. The party $\mathbf{P_1}$ computes the Cmt function for commitment generation, the party $\mathbf{P_2}$ computes the Open function for commitment verification during the protocols execution.

Note that the HMAC [19] can be used as a commitment. Then for $m \in \{0,1\}^*$ the commitment scheme is defined at Figure 4.

$\mathsf{Cmt}(m)$	$\mathsf{Open}(comm, m, op)$
1 : $op \xleftarrow{\mathcal{U}} \{0,1\}^\kappa$	1 : $comm' \leftarrow \mathsf{HMAC}(op, m)$
2 : $comm \leftarrow \mathsf{HMAC}(op, m)$	2 : **if** $(comm' \neq comm)$: **return** 0
3 : **return** $(op, comm)$	3 : **return** 1

Figure 4: HMAC as a commitment.

Key generation protocol. The party $\mathbf{P_1}$ and party $\mathbf{P_2}$ execute the KGen protocol. As a result of executing, a new key pair (d, Q) for the GOST scheme is implicitly formed. But the signing key d does not appear on either side. The output of the party $\mathbf{P_1}$ is a private key share d_1 and signature verification key Q. The output of the party $\mathbf{P_2}$ is a private key share d_2 and signature verification key Q. Note that $d = d_1 + d_2$.

A detailed description of the protocol is presented at Figure 5.

KGen

$\mathbf{P_1}$ ( )		$\mathbf{P_2}$ ( )
$d_1 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$		
$Q_1 \leftarrow d_1 \cdot P$		
$(op_Q, comm_Q) \leftarrow \mathsf{Cmt}(Q_1)$		
	$\xrightarrow{comm_Q}$	
		$d_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
		$Q_2 \leftarrow d_2 \cdot P$
	$\xleftarrow{Q_2}$	
if $(Q_2 = -Q_1)$: **return** $\perp$		
$Q \leftarrow Q_1 + Q_2$		
	$\xrightarrow{op_Q, Q_1}$	
		if $(\mathsf{Open}(comm_Q, Q_1, op_Q) = 0)$: **return** $\perp$
		if $(Q_1 = -Q_2)$: **return** $\perp$
		$Q \leftarrow Q_1 + Q_2$
return (d_1, Q)		**return** (d_2, Q)

Figure 5: Key generation protocol of the 2p-GOST signature scheme.

Verification algorithm. This algorithm can be executed by anyone with the use of verification key Q and is the same as in the GOST signature scheme. A detailed description of the algorithm is presented at Figure 6.

$\mathsf{Verify}(Q, m, \langle r, s \rangle)$

1 : **if** $(s = 0 \vee r = 0)$: **return** 0
2 : $e \leftarrow \mathrm{H}(m)$
3 : **if** $e = 0 : e \leftarrow 1$
4 : $R \leftarrow e^{-1}sP - e^{-1}rQ$
5 : **if** $(R.x \bmod q \neq r)$: **return** 0
6 : **return** 1

Figure 6: Verification algorithm of the 2p-GOST signature scheme.

Signing protocol. The party $\mathbf{P_1}$ and party $\mathbf{P_2}$ execute this protocol. Party $\mathbf{P_1}$ takes d_1, Q generated as a result of KGen and the message m as an input. Party $\mathbf{P_2}$ takes d_2, Q generated as a result of KGen and the message m as an input. As a result of executing, each of the parties receives a signature σ for the message m corresponding to the signature verification key Q.

A detailed description of the protocol is presented at Figure 7.

Sign

$\mathbf{P_1}$ (d_1, Q, m)		$\mathbf{P_2}$ (d_2, Q, m)
$k_1 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$		
$R_1 \leftarrow k_1 \cdot P$		
$(op_R, comm_R) \leftarrow \mathsf{Cmt}(R_1)$		
	$\xrightarrow{comm_R}$	
		$k_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
		$R_2 \leftarrow k_2 \cdot P$
	$\xleftarrow{R_2}$	
if $(R_2 = -R_1)$: **return** $\perp$		
$r \leftarrow (R_1 + R_2).x \mod q$		
if $(r = 0)$: **return** $\perp$		
$e \leftarrow \mathrm{H}(m)$		
if $(e = 0)$: $e \leftarrow 1$		
$s_1 \leftarrow k_1 \cdot e + d_1 \cdot r$		
	$\xrightarrow{op_R, R_1, s_1}$	
		if $(\mathsf{Open}(comm_R, R_1, op_R) = 0)$: **return** $\perp$
		if $(R_1 = -R_2)$: **return** $\perp$
		$e \leftarrow \mathrm{H}(m)$
		if $(e = 0)$: $e \leftarrow 1$
		$r \leftarrow (R_1 + R_2).x \mod q$
		$s_2 \leftarrow k_2 \cdot e + d_2 \cdot r$
		$s \leftarrow s_1 + s_2$
		$\sigma \leftarrow (r, s)$
		if $(\mathsf{Verify}(Q, m, \sigma) = 0)$: **return** $\perp$
	$\xleftarrow{s_2}$	
$s \leftarrow s_1 + s_2$		
$\sigma \leftarrow (r, s)$		
if $(\mathsf{Verify}(Q, m, \sigma) = 0)$: **return** $\perp$		
return σ		**return** σ

Figure 7: Signing protocol of the 2p-GOST signature scheme.

Note that GOST signature algorithm checks non-equality of r and s values to zero. The Sign protocol does not contain an explicit check of s value being zero, since parties

execute the Verify algorithm at the same round at which they calculate s, and the Verify algorithm contains this check. By the same reason, the party $\mathbf{P_2}$ does not check equality of r to the zero.

5 Security notions and bounds

We introduce sOMUF-PCA notion (strong One More Unforgeability under Party Compromised Attack) to analyze the security of the 2p-GOST scheme. It is a natural commonly used model [17], [15] implying that the adversary acts as one of the parties.

We prove the security under some idealized assumptions:

- we model commitment scheme as a random oracle; the commitment for KGen protocol is modeled as oracle qRO; the commitment for Sign protocol is modeled as oracle rRO. The random oracle [3] is an ideal primitive which models a random function via oracle. It provides a random output for each new query, identical input queries are given the same answer;
- we model conversion function $r = f(R)$ in the GOST signature scheme using the bijective random oracle (see detailes below). The bijective random oracle [6] is an idealized public bijection that is accessible, in both directions, via oracles.

The security is reduced to the security of the GOST scheme regarding the sUF-KO (strong Unforgeability under Key Only attack) notion and the signum-relative collision resistant property of the used hash function family.

Before proceeding to the formulation of the result, let's define the considered target security model, the signum-relative collision resistant property, the sUF-KO notion and the bijective random oracle.

sOMUF-PCA notion. Let's describe the sOMUF-PCA notion informally. The adversary $\mathcal{A}$ compromises one of the parties and communicates with the other party in the 2p-SS signature scheme. At the beginning, it can execute the KGen protocol once by querying a $KGen$ oracle. This oracle models the actions of the honest (uncompromised) party. After executing the KGen protocol adversary can execute the Sign protocol. Meanwhile, the adversary can open the parallel sessions of this protocol. For these, the adversary can make queries to the $NewSign$ oracle for opening the session and then to the $Sign$ oracle for execution the signing protocol. $Sign$ oracle models the actions of the honest party. The adversary has the capability not to finish the sessions and provoke the failures on the honest party side. The adversary's goal is to make $l+1$ correct (message, signature) pairs after l successful interactions with the honest party. The probability of achieving the goal by the adversary $\mathcal{A}$ is denoted by $\mathsf{Adv}^{\mathsf{sOMUF\text{-}PCA}}_{\mathsf{2p\text{-}GOST}}(\mathcal{A})$.

Note that such way to formulate the threat via one-more forgery captures the intuition that it is impossible to create a forgery without interacting with an honest party. It was introduced for defining unforgeability of blind signature schemes [18]. The classical way to define unforgeability for standard signature scheme is to make only one forgery that is correct and non-trivial, i.e. was not obtained as a result of honest execution of the protocol. However, in case of two-party schemes some problems may occur while defining non-triviality. Indeed, as soon as the proposed model allows the adversary not to finish the sessions, the following situation is possible. The adversary acting as $\mathbf{P_2}$ computes the signature value and does not send it to the honest party at the last flow of the signing

protocol. In this case the honest party could not determine whether the signature, returned as a forgery, is indeed fresh or was generated in the unfinished session. To address this problem we use one-more setting and consider the interaction successful if the honest party completes the computation of it's part of the signature and sends it to the adversary.

The formal description of the sOMUF-PCA notion is given in Appendix B.

Signum-relative collision resistant property. This property for a hash function family means that it is difficult to find two different messages m_1, m_2 such that the hash function values from these messages match up to the sign.

Throughout the paper we consider implicitly keyed hash functions $\mathrm{H}\colon \{0,1\}^* \mapsto \{0,1\}^h$ 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 1 (SCR property)**.** *For the family of hash functions* H

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

sUF-KO notion. Consider the sUF-KO (strong Unforgeability under Key Only attack) notion for the signature scheme SS. The adversary $\mathcal{A}$ receives signature verification key Q. It's goal is to make a forgery.

Definition 2. *For a signature scheme* SS

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

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

$\mathbf{Exp}^{\mathsf{sUF\text{-}KO}}_{\mathsf{SS}}(\mathcal{A})$

1: $(d, Q) \leftarrow \mathsf{SS.KGen}(\,)$
2: $(m, \sigma) \xleftarrow{\$} \mathcal{A}(Q)$
3: $res \leftarrow \mathsf{SS.Verify}(Q, m, \sigma)$
4: **return** res

Bijective random oracle. Bijective random oracle model (BRO model) was proposed in [6] to achieve provable security for signature schemes based on the ElGamal signature equation. In particular, GOST scheme is proven secure in the BRO model [5] (under some assumptions on the used primitives).

Bijective random oracle is used to model the mapping from group elements to the space $\mathbb{Z}_q$ used in GOST signature: $r = f(R) = R.x \bmod q$. We decompose the conversion function f as follows:

$$f = \psi \circ \Pi \circ \phi,$$

where Π is a bijection. The idea is to reflect in ϕ the structure of f that involves only its domain and to reflect in ψ the structure that involves only its range; the component that is responsible for disrupting any algebraic link between the domain and the range is modeled by Π. In security proofs we will replace Π by a bijective random oracle.

For the 2p-GOST and GOST signature schemes:

– $\phi : \mathbb{G} \to \{0,1\}^N, N = \lceil \log_2 p \rceil$, is deterministic encoding function that is implemented as the mapping the point with coordinates (x, y) to the bit representation of the x-coordinate. This is semi-injection function, i.e. it is injective except for the mutually inverse elements A, B for which the equality $\phi(A) = \phi(B)$ holds;

– $\psi : \{0, 1, \ldots, 2^N - 1\} \to \mathbb{Z}_q$ is a function that maps integer to elements of $\mathbb{Z}_q$ that is implemented as the reduction of an integer modulo q;

– $\Pi : \{0,1\}^N \to \{0, 1, \ldots, 2^N - 1\}$ is the link in the middle, bridging the range of ϕ with the domain of ψ.

Finally, we are ready to formulate the security bound for the 2p-GOST scheme.

Theorem 1. *Let $\mathcal{A}$ be an adversary with time complexity T in the* sOMUF-PCA *model for the* 2p-GOST *scheme, making at most q_R and q_Q queries to the random oracles rRO and qRO respectively, at most q_{BRO} and $q_{BRO^{-1}}$ queries to the bijective random oracles BRO and BRO^{-1} respectively and at most q_{sign} queries to the oracle $NewSign$. Then, there exist an adversary $\mathcal{B}$ in the* sUF-KO *model for the* GOST *scheme and an adversary $\mathcal{C}$ that breaks the signum-relative collision resistant property of* H*, such that:*

$$\mathsf{Adv}^{\mathsf{sOMUF\text{-}PCA}}_{\mathsf{2p\text{-}GOST}}(\mathcal{A}) \leqslant \mathsf{Adv}^{\mathsf{sUF\text{-}KO}}_{\mathsf{GOST}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{SCR}}_{\mathsf{H}}(\mathcal{C}) + \\ + \frac{q_Q + q_{sign} \cdot (q_R + q_{sign})}{2^{\min\{\kappa, n\}}} + \frac{2(q_{BRO} + q_{BRO^{-1}} + 3q_{sign} + 1)^2}{q},$$

where κ, n are the parameters of the underlying commitment scheme.

An adversary $\mathcal{B}$ makes at most $(q_{BRO} + 2q_{sign} + 1)$ and $q_{BRO^{-1}}$ queries to the bijective random oracles BRO and BRO^{-1} respectively. The time complexities of $\mathcal{B}$ and $\mathcal{C}$ are at most $3T$.

The proof of the theorem is provided in Appendix C.

The interpretation of the random oracle model and bijective random oracle model in our case is as follows. We do not cover the methods of cryptoanalysis that use the features of structure of the concrete commitment scheme to link its domain and range or exploit the connection between two algebraic structures: bit strings encoding the coordinates of elliptic curve points and the corresponding integers (see [3], [6]).

Let discuss the obtained security bound. Each term of the bound corresponds to the specific directions of cryptoanalysis that are meaningful for the proposed scheme. The first two terms reflect methods targeted at breaking the security of the underlying cryptographic mechanisms — the GOST signature scheme (in the no-message setting) and the hash function. Obviously, breaking each of these mechanisms allows to obtain a forgery for 2p-GOST.

The third term reflects methods of cryptoanalysis targeted at the commitment scheme (as a black box) and assuming the dishonest computation of commitment values by one of the parties. Indeed, this term is equal to the probability of guessing the input (output) of the commitment function, modelled as a random oracle, by its output (input) without querying it. Note that if the adversary is able to do so, the attacks described in Section 3 for the naive version of 2p-GOST become possible.

The last term in the bound reflects methods of cryptoanalysis assuming gathering the large number of (message, signature) pairs and exploiting some collisions or other connections of their values. A prime example of such attack is to find two signatures

generated with the same $k = k_1 + k_2$ value and recovering the signing key. The success of such attack is of order $\frac{q_{sign}^2}{q}$.

Note that the obtained security bound demonstrates that our method of constructing two-party scheme based on the GOST scheme does not add any additional security assumptions except for the assumption that commitment is modelled as random oracle. Indeed, other two assumptions, bijective random oracle and signum-relative collision resistance of the used hash function family, are also the underlying assumptions for the security of the GOST scheme in the chosen-message setting (for details see [5]).

6 Conclusion

The first result of this paper is devoted to the analysis of existing signature schemes based on GOST signature equation, in which the signature is generated by several signers. We show that all these schemes are not suitable for providing signing key protection on the user mobile device. Some of these schemes use a trusted third party, others are proven secure in the weak security models. Moreover, we provide the attacks breaking unforgeability for some of these schemes.

The second result of this paper is devoted to the synthesis of the secure two-party signature scheme with the same verification algorithm as in the GOST signature scheme. We propose the 2p-GOST scheme which uses the commitment scheme. We prove that this scheme is secure in the case when one of the parties is malicious under the assumption that the classical GOST scheme is unforgeable and commitment scheme is modelled as a random oracle. This scheme can be used for providing signing key protection on the user mobile device.

References

[1] Beresneva, Anastasia and Epishkina, Anna and Isupova, Olga and Kogos, Konstantin and Shimkiv, Mikhail, "Special digital signature schemes based on GOST R 34.10-2012", *IEEE*, 2016, 135–140.

[2] Benhamouda, Fabrice and Lepoint, Tancrède and Loss, Julian and Orrù, Michele and Raykova, Mariana, "On the (in) security of ROS", *Journal of Cryptology*, **35**:4 (2022), 25.

[3] Bellare, Mihir and Rogaway, Phillip, "Random oracles are practical: A paradigm for designing efficient protocols", *Proceedings of the 1st ACM Conference on Computer and Communications Security*, 1993, 62–73.

[4] Dzhunkovsky P. O., Ditenkova A. S., "Porogovaya schema podpisi s razdeleniem secreta na baze GOST R 34.10-2001 [Threshold scheme of a digital signature with a shared secret based on GOST R 34.10-2001]", *Bezopasnost' Informatsionnykh Tekhnologiy [IT Security (Russia)],*, **17**:3 (2010), 61–65.

[5] Fersch, Manuel, "The provable security of elgamal-type signature schemes", 2018.

[6] Fersch, Manuel and Kiltz, Eike and Poettering, Bertram, "On the provable security of (EC) DSA signatures", *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, 1651–1662.

[7] Gennaro, Rosario and Jarecki, Stanislaw and Krawczyk, Hugo and Rabin, Tal, "Secure distributed key generation for discrete-log based cryptosystems", *Journal of Cryptology*, **20** (2007), 51–83.

[8] *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.

[9] *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.

[10] *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.

[11] 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.

[12] Komlo, Chelsea and Goldberg, Ian, "FROST: flexible round-optimized Schnorr threshold signatures", *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27*, Springer, 2021, 34–65.

[13] Kim, Sungwook and Kim, Jihye and Cheon, Jung Hee and Ju, Seong-ho, "Threshold signature schemes for ElGamal variants", *Computer Standards & Interfaces*, **33**:4 (2011), 432–437.

[14] Kim, Tuan Nguyen and Ngoc, Duy Ho and Moldovyan, Nikolay A, "New Collective Signatures Based on the Elliptic Curve Discrete Logarithm Problem", *CMC-COMPUTERS MATERIALS & CONTINUA*, **73**:1 (2022), 595–610.

[15] Lindell, Yehuda, "Fast secure two-party ECDSA signing", *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part II 37*, Springer, 2017, 613–644.

[16] Moldovyan, Nikolai Andreevich, "Theoretical minimum and digital signature algorithms [there is a vulture]", 2010.

[17] Nicolosi, Antonio and Krohn, Maxwell N and Dodis, Yevgeniy and Mazieres, David, "Proactive Two-Party Signatures for User Authentication,", *NDSS*, 2003.

[18] Pointcheval, David and Stern, Jacques, "Provably secure blind signature schemes", *Advances in Cryptology—ASIACRYPT'96: International Conference on the Theory and Applications of Cryptology and Information Security Kyongju, Korea, November 3–7, 1996 Proceedings*, Springer, 1996, 252–265.

[19] Krawczyk H., Bellare M., Canetti R., *HMAC: Keyed-hashing for message authentication*, RFC2104, 1997, https://www.rfc-editor.org/info/rfc2104.

[20] Zhang, Yunru and Luo, Min and Choo, Kim-Kwang Raymond and Li, Li and He, Debiao, "Efficient and Secure Two-Party Distributed Signing Protocol for the GOST Signature Algorithm", *Security and Privacy in Social Networks and Big Data: 6th International Symposium, SocialSec 2020, Tianjin, China, September 26–27, 2020, Proceedings 6*, Springer, 2020, 3–19.

A ROS-style attacks

A.1 Scheme Zhang-Luo-Choo-Li-He

The scheme proposed in [20] uses the additively homomorphic encryption scheme $(Enc_{\mathsf{pk}}(\cdot), Dec_{\mathsf{sk}}(\cdot))$, where keys $(\mathsf{sk}, \mathsf{pk})$ are known to $\mathbf{P_1}$, and $c_d = Enc_{\mathsf{pk}}(d_1)$ is known to $\mathbf{P_2}$. In the attack, we use encryption scheme honestly, so details related to its correct using are omitted.

The signing protocol of this scheme is presented at Figure 8.

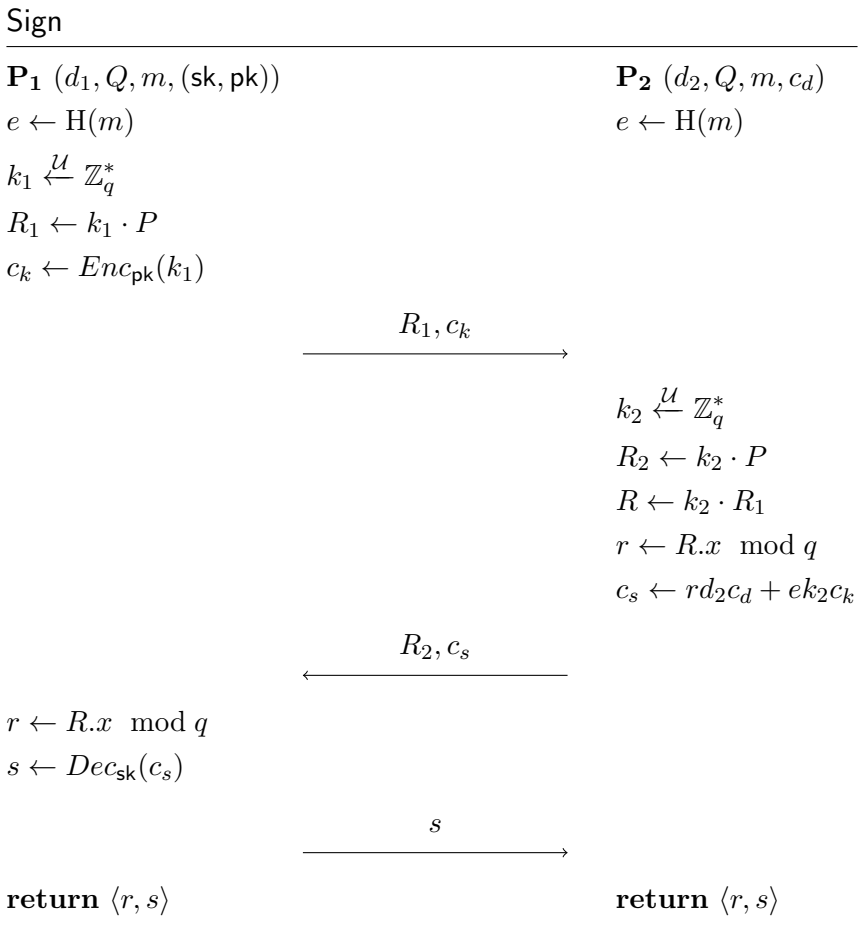

Figure 8: Signing protocol of the signature scheme [20].

The attack, presented below, allows an adversary acting as $\mathbf{P_2}$ to construct $(l+1)$ correct (message, signature) pairs after $l \geqslant \lceil \log q \rceil$ successful interactions with $\mathbf{P_1}$. The adversary acts as follows:

1. Selects message $m_l \in \{0,1\}^*$ for which a signature will be forged, let $e_l = \mathrm{H}(m_l)$.
2. Opens l parallel sessions for some messages $m_0, \ldots, m_{l-1}$, querying $\mathbf{P_1}$, let $e_i = \mathrm{H}(m_i), 0 \leqslant i \leqslant l-1$, and receives corresponding points $R_1^0, \ldots, R_1^{l-1}$.
3. Selects $k_2^{i,0}, k_2^{i,1} \in \mathbb{Z}_q^*, 0 \leqslant i \leqslant l-1$, then $R^{i,0} = k_2^{i,0} R_1^i$, $R^{i,1} = k_2^{i,1} R_1^i$, $0 \leqslant i \leqslant l-1$, $r_{i,0} = R^{i,0}.x \mod q$, $r_{i,1} = R^{i,1}.x \mod q$, $0 \leqslant i \leqslant l-1$, such that $k_2^{i,1^{-1}} r_{i,1} \neq k_2^{i,0^{-1}} r_{i,0}$, $0 \leqslant i \leqslant l-1$.

4. Defines $(\rho_0, \rho_1, \ldots, \rho_l)$ as the vector of coefficients placed before x_i in the function $f : \mathbb{Z}_q^l \to \mathbb{Z}_q$; $f(x_0, \ldots, x_{l-1}) = \sum_{i=0}^{l-1} 2^i \underbrace{\frac{x_i - {k_2^{i,0}}^{-1} r_{i,0}}{{k_2^{i,1}}^{-1} r_{i,1} - {k_2^{i,0}}^{-1} r_{i,0}}}_{b'_i} = \sum_{i=0}^{l-1} \rho_i x_i + \rho_l$. Note that if $x_i = {k_2^{i,0}}^{-1} r_{i,0}$ then $b'_i = 0$, if $x_i = {k_2^{i,1}}^{-1} r_{i,1}$ then $b'_i = 1$.
5. Defines $R^l = e_l^{-1} \left(\sum_{i=0}^{l-1} \rho_i e_i R_1^i - \rho_l Q \right)$.
6. Defines $r_l = R^l.x \mod q$.
7. Defines $b_0, \ldots, b_{l-1}$ from the following equation: $r_l = \sum_{i=0}^{l-1} 2^i b_i$.
8. Defines $k_2^i = k_2^{i,b_i}, r_i = r_{i,b_i}, 0 \leqslant i \leqslant l-1$; therefore, according to step 4, $r_l = \sum_{i=0}^{l-1} 2^i b_i = \sum_{i=0}^{l-1} \rho_i {k_2^i}^{-1} r_i + \rho_l$.
9. Defines $R_2^i = k_2^i P$, $0 \leqslant i \leqslant l-1$.
10. Calculates $c_s^0, \ldots, c_s^{l-1}$, according to the protocol.
11. Sends $R_2^0, \ldots, R_2^{l-1}$ and $c_s^0, \ldots, c_s^{l-1}$ values to $\mathbf{P_1}$ in the corresponding sessions;
12. Obtains responses $s^0, \ldots, s^{l-1}$ such that:
$$ {k_2^i}^{-1} s^i = k_1^i e_i + d_1 d_2 r_i {k_2^i}^{-1}, \ 0 \leqslant i \leqslant l-1. $$
13. Defines $s^l = \sum_{i=0}^{l-1} \rho_i {k_2^i}^{-1} s^i = \sum_{i=0}^{l-1} \rho_i k_1^i e_i + \sum_{i=0}^{l-1} \rho_i d_1 d_2 r_i {k_2^i}^{-1}$.
14. Outputs $\{m_i, (r_i, s^i)\}_{i=0}^l$.

Indeed, for $0 \leqslant i \leqslant l-1$ signature (r_i, s^i) is valid for m_i by attack construction. Consider the case $i = l$.

We show that the following signature verification equation holds:

$$ {e_l}^{-1} s^l P = R^l + {e_l}^{-1} r_l Q. $$

The left side:

$$ {e_l}^{-1} s^l P = {e_l}^{-1} \sum_{i=0}^{l-1} \rho_i e_i R_1^i + Q {e_l}^{-1} \sum_{i=0}^{l-1} \rho_i r_i {k_2^i}^{-1}. $$

The right side:

$$ R^l + {e_l}^{-1} r_l Q = e_l^{-1} \left(\sum_{i=0}^{l-1} \rho_i e_i R_1^i - \rho_l Q \right) + {e_l}^{-1} Q \left(\sum_{i=0}^{l-1} \rho_i {k_2^i}^{-1} r_i + \rho_l \right) = $$
$$ = {e_l}^{-1} \sum_{i=0}^{l-1} \rho_i e_i R_1^i + Q {e_l}^{-1} \sum_{i=0}^{l-1} \rho_i r_i {k_2^i}^{-1}. $$

A condition $l \geqslant \lceil \log q \rceil$ is is necessary in order to be able to carry out the step 7.

A.2 ROS attack on the straightforward scheme

This section contains some modification of the attack, described in Appendix A.1, for scheme defined at Figure 2.

We describe only the different steps:

3. Selects $k_2^{i,0}, k_2^{i,1} \in \mathbb{Z}_q^*, 0 \leqslant i \leqslant l-1$, then $R^{i,0} = k_2^{i,0}P + R_1^i$, $R^{i,1} = k_2^{i,1}P + R_1^i$, $0 \leqslant i \leqslant l-1$, $r_{i,0} = R^{i,0}.x \mod q$, $r_{i,1} = R^{i,1}.x \mod q$, $0 \leqslant i \leqslant l-1$, such that $r_{i,1}e_l(e_i)^{-1} \neq r_{i,0}e_l(e_i)^{-1}$, $0 \leqslant i \leqslant l-1$.

4. Defines $(\rho_0, \rho_1, \ldots, \rho_l)$ as the vector of coefficients placed before x_i in the function $f : \mathbb{Z}_q^l \to \mathbb{Z}_q$; $f(x_0, \ldots, x_{l-1}) = \sum_{i=0}^{l-1} 2^i \underbrace{\frac{x_i - r_{i,0}e_l(e_i)^{-1}}{r_{i,1}e_l(e_i)^{-1} - r_{i,0}e_l(e_i)^{-1}}}_{b_i'} = \sum_{i=0}^{l-1} \rho_i x_i + \rho_l$. Note that if $x_i = r_{i,0}e_l(e_i)^{-1}$ then $b_i' = 0$, if $x_i = r_{i,1}e_l(e_i)^{-1}$ then $b_i' = 1$.

5. Defines $R_1^l = \sum_{i=0}^{l-1} \rho_i R_1^i - e_l^{-1}\rho_l Q_1$.

6. Selects $k_2^l \in \mathbb{Z}_q^*$ and defines $R_2^l = k_2^l P$. Defines $R^l = R_1^l + R_2^l$ and $r_l = R^l.x \mod q$.

8. Defines $k_2^i = k_2^{i,b_i}, r_i = r_{i,b_i}, 0 \leqslant i \leqslant l-1$; $r_l = \sum_{i=0}^{l-1} 2^i b_i = e_l \sum_{i=0}^{l-1} \rho_i e_i^{-1} r_i + \rho_l$.

10. Calculates $s_2^0, \ldots, s_2^{l-1}$, according to the protocol.

11. Sends $R_2^0, \ldots, R_2^{l-1}$ and $s_2^0, \ldots, s_2^{l-1}$ values to $\mathbf{P_1}$ in the corresponding sessions.

12. Obtains responses $s_1^0, \ldots, s_1^{l-1}$ such that:

$$s_1^i = k_1^i e_i + d_1 r_i \; 0 \leqslant i \leqslant l-1.$$

13 Defines $s_1^l = e_l \sum_{i=0}^{l-1} \rho_i {e_i}^{-1} s_1^i$, $s_2^l = k_2^l e_l + r_l d_2$, $s^i = s_1^i + s_2^i; 0 \leqslant i \leqslant l$.

Indeed, for $0 \leqslant i \leqslant l-1$ signature (r_i, s^i) is valid for m_i by attack construction. Consider the case $i = l$.

We show that the following signature verification equation holds:

$$R^l = {e_l}^{-1}(s^l P - r_l Q).$$

$$
\begin{aligned}
{e_l}^{-1}(s^l P - r_l Q) = {e_l}^{-1}(s_1^l P + s_2^l P - r_l Q_1 - r_l Q_2) = \\
= \sum_{i=0}^{l-1} \rho_i {e_i}^{-1} s_1^i P + k_2^l P + e_l^{-1} r_l d_2 P - e_l^{-1} r_l Q_2 - e_l^{-1} r_l Q_1 = \\
= \sum_{i=0}^{l-1} \rho_i {e_i}^{-1} s_1^i P - e_l^{-1} r_l Q_1 + R_2^l = \\
= \sum_{i=0}^{l-1} \rho_i {e_i}^{-1} s_1^i P - Q_1 \left(\sum_{i=0}^{l-1} \rho_i e_i^{-1} r_i + e_l^{-1} \rho_l \right) + R_2^l = \\
= \sum_{i=0}^{l-1} \rho_i \underbrace{e_i^{-1}(s_1^i P - r_i Q_1)}_{=R_1^i} - \rho_l e_l^{-1} Q_1 + R_2^l = \\
= \underbrace{\sum_{i=0}^{l-1} \rho_i R_1^i - \rho_l e_l^{-1} Q_1}_{=R_1^l} + R_2^l = \\
= R_1^l + R_2^l = R^l.
\end{aligned}
$$

A.3 ROS attack on the scheme with sent message

Let's describe a ROS attack on the following modification of the singing protocol at Figure 2: the message m is argument only for $\mathbf{P_1}$, the party $\mathbf{P_2}$ receives m from the party $\mathbf{P_1}$ in the third transmission.

This attack uses an opportunity to open several parallel sessions. The attack allows an adversary acting as $\mathbf{P_1}$ to construct $(l+1)$ correct (message, signature) pairs after $l \geqslant \lceil \log q \rceil$ successful interactions with $\mathbf{P_2}$.

The adversary acts as follows:

1. Selects message $m_l \in \{0,1\}^*$ for which a signature will be forged, let $e_l = \mathrm{H}(m_l)$.
2. Opens l parallel sessions, selects $R_1^i = k_1^i P,\ 0 \leqslant i \leqslant l-1$, and sends corresponding $comm_R^0, \ldots, comm_R^{l-1}$ to the second user. Receives $R_2^0, \ldots, R_2^{l-1}$.
3. Defines $r_i = (R_1^i + R_2^i).x \mod q, 0 \leqslant i \leqslant l-1$.
4. Selects $m_i^0, m_i^1, 0 \leqslant i \leqslant l-1$, such that $r'_{i,0} \neq r'_{i,1}$, where:
$$e_i^0 = \mathrm{H}(m_i^0),\ e_i^1 = \mathrm{H}(m_i^1),$$
$$r'_{i,0} = e_l (e_i^0)^{-1} r_i,\ r'_{i,1} = e_l (e_i^1)^{-1} r_i.$$
5. Defines $(\rho_0, \rho_1, \ldots, \rho_l)$ as the vector of coefficients placed before x_i in the function $f : \mathbb{Z}_q^l \to \mathbb{Z}_q;\ f(x_0, \ldots, x_{l-1}) = \sum_{i=0}^{l-1} \underbrace{2^i \frac{x_i - r'_{i,0}}{r'_{i,1} - r'_{i,0}}}_{b'_i} = \sum_{i=0}^{l-1} \rho_i x_i + \rho_l$. Note that if $x_i = r'_{i,0}$ then $b'_i = 0$, if $x_i = r'_{i,1}$ then $b'_i = 1$.

6. Defines $R_2^l = \sum_{i=0}^{l-1} \rho_i R_2^i - e_l^{-1} \rho_l Q_2$.

7. Selects k_1^l from $\mathbb{Z}_q^*$ and defines $R_1^l = k_1^l P$.

8. Defines $r_l = (R_1^l + R_2^l).x \mod q$.

9. Defines $b_0, \ldots, b_{l-1}$ from the following equation: $r_l = \sum_{i=0}^{l-1} 2^i b_i$.

10. Defines $r_i' = r_{i,b_i}', e_i = e_i^{b_i}, m_i = m_i^{b_i}, 0 \leqslant i \leqslant l-1$; therefore $r_l = \sum_{i=0}^{l-1} \rho_i r_i' + \rho_l = e_l \sum_{i=0}^{l-1} \rho_i e_i^{-1} r_i + \rho_l$.

11. Calculates s_1^i, according to the protocol: $s_1^i = k_1^i \cdot e_i + r_i \cdot d_1$.

12. Sends $op_R^i, R_1^i, s_1^i,\ 0 \leqslant i \leqslant l-1$, values to $\mathbf{P_2}$ in the corresponding opened sessions.

13. Obtains responses $s_2^0, \ldots, s_2^{l-1}$such that:

$$s_2^i P = e_i R_2^i + r_i Q_2,\ 0 \leqslant i \leqslant l-1.$$

14. Defines $s_2^l = e_l \sum_{i=0}^{l-1} \rho_i e_i^{-1} s_2^i$. Calculates $s_1^l = k_1^l e_l + r_l \cdot d_1$.

15. Defines $s^i = s_1^i + s_2^i, 0 \leqslant i \leqslant l$.

16. Outputs $\{m_i, (r_i, s^i)\}_{i=0}^l$.

Indeed, for $0 \leqslant i \leqslant l-1$ signature (r_i, s^i) is valid for m_i by attack construction. Consider the case $i = l$.

We show that the following signature verification equation holds:

$$R^l = {e_l}^{-1}(s^l P - r_l Q).$$

$$R^{l'} = {e_l}^{-1}\left(s^l P - r_l Q\right) = {e_l}^{-1}\left((s_1^l + s_2^l)P - r_l(Q_1 + Q_2)\right) =$$

$$= {e_l}^{-1}\left(e_l \sum_{i=0}^{l-1} \rho_i e_i^{-1} s_2^i P - \left(e_l \sum_{i=0}^{l-1} \rho_i e_i^{-1} r_i + \rho_l\right) \cdot Q_2 + \underbrace{(s_1^l P - r_l Q_1)}_{=e_l R_1^l}\right) =$$

$$= \sum_{i=0}^{l-1} \rho_i e_i^{-1} s_2^i P - \sum_{i=0}^{l-1} \rho_i e_i^{-1} r_i Q_2 - {e_l}^{-1}\rho_l Q_2 + R_1^l =$$

$$= \sum_{i=0}^{l-1} \rho_i \underbrace{e_i^{-1}(s_2^i P - r_i Q_2)}_{=R_2^i} - {e_l}^{-1}\rho_l Q_2 + R_1^l = \underbrace{\sum_{i=0}^{l-1} \rho_i R_2^i - {e_l}^{-1}\rho_l Q_2}_{=R_2^l} + R_1^l = R_2^l + R_1^l.$$

and $R^{l'}.x = r_l \mod q$ from the step 8.

A condition $l \geqslant \lceil \log q \rceil$ is is necessary in order to be able to carry out the step 9.

B Security notions

In this section we formally define the security model used for two-party signature schemes.

Definition 3. *For a two-party signature scheme* 2p-SS

$$\mathsf{Adv}^{\mathsf{sOMUF\text{-}PCA}}_{\mathsf{2p\text{-}SS}}(\mathcal{A}) = \Pr\big[\mathbf{Exp}^{\mathsf{sOMUF\text{-}PCA}}_{\mathsf{2p\text{-}SS}}(\mathcal{A}) \to 1\big],$$

where the experiment $\mathbf{Exp}^{\mathsf{sOMUF\text{-}PCA}}_{\mathsf{2p\text{-}SS}}(\mathcal{A})$ *is defined in the following way:*

$\underline{\mathbf{Exp}^{\mathsf{sOMUF\text{-}PCA}}_{\mathsf{2p\text{-}SS}}(\mathcal{A})}$
$\Pi \xleftarrow{\mathcal{U}} Perm\left(\{0,1\}^N \to \{0,\dots,2^N-1\}\right)$
$\Pi_R \xleftarrow{\mathcal{U}} Func(\{0,1\}^\kappa \times \mathbb{G} \to \{0,1\}^n)$
$\Pi_Q \xleftarrow{\mathcal{U}} Func(\{0,1\}^\kappa \times \mathbb{G} \to \{0,1\}^n)$
$l \leftarrow 0,\ SESS \leftarrow \emptyset$
$sid \leftarrow -1$
$round_{kg} \leftarrow 0,\ ctx_{kg} \leftarrow \emptyset$
$p \leftarrow \mathcal{A}()$
if $(p \neq 1 \wedge p \neq 2)$: **return** $\perp$
$(Q, d_p) \leftarrow (\varepsilon, \varepsilon)$
$\{(m_i, \langle r_i, s_i\rangle)\}_{i=1}^{l+1} \xleftarrow{\$} \mathcal{A}^{KGen, NewSign, Sign, BRO, BRO^{-1}, rRO, qRO}(p)$
return $((\forall i \neq j \in \{1,\dots,l+1\}:$
$(m_i, \langle r_i, s_i\rangle) \neq (m_j, \langle r_j, s_j\rangle)) \wedge$
$\wedge\ (\forall i \in \{1,\dots,l+1\} : \mathsf{Verify}(Q, m_i, \langle r_i, s_i\rangle)))$

$\underline{Oracle\ BRO(\alpha)}$
return $\Pi(\alpha)$

$\underline{Oracle\ BRO^{-1}(\beta)}$
return $\Pi^{-1}(\beta)$

$\underline{rRO(op_R, R)}$
return $\Pi_R(op_R, R)$

$\underline{qRO(op_Q, Q)}$
return $\Pi_Q(op_Q, Q)$

$\underline{KGen(msg)}$
if $(Q \neq \varepsilon)$: **return** $\perp$
return $\mathsf{ExecKGen}^p(msg)$

$\underline{NewSign(m)}$
if $(Q = \varepsilon \vee d_p = \varepsilon)$: **return** $\perp$
$round \leftarrow 0,\ ctx \leftarrow \{round\},\ flag \leftarrow 0$
$state \leftarrow (m, ctx, flag)$
$sid \leftarrow sid + 1$
$SESS \leftarrow SESS \cup \{(sid, state)\}$
return sid

$\underline{Sign(sid, msg)}$
if $((sid, \cdot) \notin SESS)$: **return** $\perp$
$state \leftarrow SESS[sid]$
$(state', msg') \leftarrow \mathsf{ExecSign}^p(state, msg)$
if $(msg' = \perp)$: **return** $\perp$
$SESS[sid] \leftarrow state'$
$(m, ctx, flag) \leftarrow state'$
if $(flag)$: $l \leftarrow l + 1$
return msg'

where $\mathsf{ExecKGen}^p$ *and* $\mathsf{ExecSign}^p$ *are functions that define the execution of the* KGen *and* Sign *protocols of the* 2p-SS *scheme by an uncompromised party, i.e.* P_{3-p}.

Let's define the functions $\mathsf{ExecKGen}^p$ and $\mathsf{ExecSign}^p$, where $p = 1, 2$, for the 2p-GOST scheme.

$\mathsf{ExecKGen}^1(msg)$

1 : **if** $(round_{kg} = 0)$:
2 : $\quad d_1 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
3 : $\quad Q_1 \leftarrow d_1 P$
4 : $\quad op_Q \xleftarrow{\mathcal{U}} \{0,1\}^\kappa$
5 : $\quad comm_Q \leftarrow qRO(op_Q, Q_1)$
6 : $\quad msg' \leftarrow \{comm_Q\}$
7 : **else if** $(round_{kg} = 1)$:
8 : $\quad Q_2 \leftarrow msg$
9 : $\quad$ **if** $(Q_2 = -Q_1)$: **return** $\perp$
10 : $\quad Q \leftarrow Q_1 + Q_2$
11 : $\quad msg' \leftarrow \{op_Q, Q_1\}$
12 : **else** :
13 : $\quad msg' \leftarrow \varepsilon$
14 : $round_{kg} \leftarrow round_{kg} + 1$
15 : $\quad$ // Update the ctx_{kg} value
16 : **return** msg'

$\mathsf{ExecSign}^1(state, msg)$

1 : $round \leftarrow state.ctx.round$
2 : **if** $(round = 0)$:
3 : $\quad e \leftarrow \mathrm{H}(state.m)$
4 : $\quad$ **if** $(e = 0) : e \leftarrow 1$
5 : $\quad k_1 \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
6 : $\quad R_1 \leftarrow k_1 P$
7 : $\quad op_R \xleftarrow{\mathcal{U}} \{0,1\}^\kappa$
8 : $\quad comm_R \leftarrow rRO(op_R, R_1)$
9 : $\quad msg' \leftarrow \{comm_R\}$
10 : **else if** $(round = 1)$:
11 : $\quad R_2 \leftarrow msg$
12 : $\quad$ **if** $(R_2 = -R_1)$:
13 : $\quad\quad$ **return** $(state, \perp)$
14 : $\quad R \leftarrow R_1 + R_2$
15 : $\quad r \leftarrow \psi(\Pi(\phi(R)))$
16 : $\quad$ **if** $(r = 0)$: **return** $(state, \perp)$
17 : $\quad s_1 \leftarrow k_1 \cdot e + d_1 \cdot r$
18 : $\quad state.flag \leftarrow 1$
19 : $\quad msg' \leftarrow \{op_R, R_1, s_1\}$
20 : **else if** $(round = 2)$:
21 : $\quad s_2 \leftarrow msg$
22 : $\quad s \leftarrow s_1 + s_2$
23 : $\quad msg' \leftarrow \varepsilon$
24 : $\quad$ **if** $(\mathsf{Verify}(Q, state.m, \langle r, s\rangle) = 0)$:
25 : $\quad\quad$ **return** $(state, \perp)$
26 : **else**
27 : $\quad msg' \leftarrow \varepsilon$
28 : $\quad$ // Update the $state.ctx$ value
29 : **return** $(state, msg')$

$\mathsf{ExecKGen}^2(msg)$

1 : **if** $(round_{kg} = 0)$:
2 : $\quad comm_Q \leftarrow msg$
3 : $\quad d_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
4 : $\quad Q_2 \leftarrow d_2 P$
5 : $\quad msg' \leftarrow \{Q_2\}$
6 : **else if** $(round_{kg} = 1)$:
7 : $\quad op_Q, Q_1 \leftarrow msg$
8 : $\quad$ **if** $(comm_Q \neq qRO(op_Q, Q_1))$:
9 : $\quad\quad$ **return** $\perp$
10 : $\quad$ **if** $(Q_1 = -Q_2)$: **return** $\perp$
11 : $\quad Q \leftarrow Q_1 + Q_2$
12 : $\quad msg' \leftarrow \varepsilon$
13 : **else** :
14 : $\quad msg' \leftarrow \varepsilon$
15 : $round_{kg} \leftarrow round_{kg} + 1$
16 : $\quad$ // Update the ctx_{kg} value
17 : **return** msg'

$\mathsf{ExecSign}^2(state, msg)$

1 : $round \leftarrow state.ctx.round$
2 : **if** $(round = 0)$:
3 : $\quad e \leftarrow \mathrm{H}(state.m)$
4 : $\quad$ **if** $(e = 0) : e \leftarrow 1$
5 : $\quad comm_R \leftarrow msg$
6 : $\quad k_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
7 : $\quad R_2 \leftarrow k_2 P$
8 : $\quad msg' \leftarrow \{R_2\}$
9 : **else if** $(round = 1)$:
10 : $\quad (op_R, R_1, s_1) \leftarrow msg$
11 : $\quad$ **if** $(comm_R \neq rRO(op_R, R_1))$:
12 : $\quad\quad$ **return** $(state, \perp)$
13 : $\quad$ **if** $(R_1 = -R_2)$:
14 : $\quad\quad$ **return** $(state, \perp)$
15 : $\quad R \leftarrow R_1 + R_2$
16 : $\quad r \leftarrow \psi(\Pi(\phi(R)))$
17 : $\quad s_2 \leftarrow k_2 \cdot e + d_2 \cdot r$
18 : $\quad s \leftarrow s_1 + s_2$
19 : $\quad state.flag \leftarrow 1$
20 : $\quad msg' \leftarrow \{s_2\}$
21 : $\quad$ **if** $(\mathsf{Verify}(Q, state.m, \langle r, s\rangle) = 0)$:
22 : $\quad\quad$ **return** $(state, \perp)$
23 : **else** :
24 : $\quad msg' \leftarrow \varepsilon$
25 : $\quad$ // Update the $state.ctx$ value
26 : **return** $(state, msg')$

C Security proof of the scheme

Proof. Let's $\mathbf{Exp}^0(\mathcal{A})$ denote the original security experiment as defined in the sOMUF-PCA security model definition (see Definition 3). We fix $\mathcal{A}$ – the adversary that makes forgery for the 2p-GOST scheme in the sOMUF-PCA model. The adversary has the access to the random oracles rRO, qRO, the bijective random oracles BRO and BRO^{-1}, the key generation oracle $KGen$, the $NewSign$ oracle, initiating a new signing session, and the signing oracle $Sign$. We assume that adversary can make at most q_R and q_Q queries to the oracles rRO and qRO respectively, at most q_{BRO} and $q_{BRO^{-1}}$ queries to the oracles BRO and BRO^{-1} respectively and at most q_{sign} queries to the oracle $NewSign$. Our goal is to upper-bound $\Pr\big[\mathbf{Exp}^{\mathsf{sOMUF\text{-}PCA}}_{\mathsf{2p\text{-}GOST}}(\mathcal{A}) \to 1\big] = \Pr\big[\mathbf{Exp}^0(\mathcal{A}) \to 1\big]$.

Construction of adversary $\mathcal{C}$. $\mathbf{Exp}^1(\mathcal{A})$ is the modification of the $\mathbf{Exp}^0(\mathcal{A})$ obtained by implementing Π, Π_R, Π_Q using «lazy sampling» (see Figure 9). Here and after we denote the difference between experiments by color in 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.

The idea is to «open» new pairs $(\alpha, \Pi(x))$ and triples $(op_R, R, \Pi_R(op_R, R))$ or $(op_Q, Q, \Pi_Q(op_Q, Q))$ as soon as the adversary asks for it. From now onward we denote by Π the subset of $(\{0,1\}^N, \{0, \ldots, 2^N - 1\})$, which is defined by the union of two sets Π^S and Π^O. We store the pairs obtained from queries to the BRO and BRO^{-1} oracles 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 $\Pi(\alpha)$ and α as $\Pi^{-1}(\beta)$. We write $(\alpha, \cdot) \in \Pi$ shorthand for the condition that there exists β such that $(\alpha, \beta) \in \Pi$. We write $(\cdot, \alpha) \in \Pi$ shorthand for the condition that there exists β such that $(\alpha, \beta) \in \Pi$. Analogically, we denote by Π_R and Π_Q the subsets of $(\{0,1\}^\kappa, \mathbb{G}, \{0,1\}^n)$, that store the triples obtained from queries to the rRO and qRO oracle respectively. The shorthands for the conditions that there exist the triples belonging to the corresponding sets are defined in the same way as for Π set.

These modifications do not affect the distribution on qRO and rRO outputs. There are the following differences between $\mathbf{Exp}^0(\mathcal{A})$ and $\mathbf{Exp}^1(\mathcal{A})$ in implementing permutation Π:

1. at the BRO oracle: **abort** if $(\cdot, \beta) \in \Pi$ (line 3);
2. at the BRO^{-1} oracle: **abort** if $(\alpha, \cdot) \in \Pi$ (line 3);
3. at the $Sign$ oracle in the function $\mathsf{ExecSign}^1$ or the function $\mathsf{ExecSign}^2$: **abort** if $(\cdot, \beta) \in \Pi$ (line 19 or 24).

To estimate the difference between $\mathbf{Exp}^0(\mathcal{A})$ and $\mathbf{Exp}^1(\mathcal{A})$, we should estimate the probability that $\mathbf{Exp}^1(\mathcal{A})$ aborts in these ways.

Let's consider the BRO oracle. Note that is executed not only when adversary makes direct query to it but also during the Verify procedure (in signing oracle and finalization of the experiment). Since the number of forgeries does not exceed $(q_{sign}+1)$, the number of BRO executions does not exceed $(q_{BRO}+2q_{sign}+1)$. The value β is uniformly distributed on a set $\{0, \ldots, 2^N-1\}$ of cardinality 2^N. In the worst case the adversary $\mathcal{A}$ has already made all queries to the BRO, BRO^{-1}, $Sign$ oracles and thus Π contains at least $(q_{BRO} + q_{BRO^{-1}} + 3q_{sign} + 1)$ elements. The abort condition is met if the value β hits one of elements in Π. We can estimate this probability as $\dfrac{q_{BRO} + q_{BRO^{-1}} + 3q_{sign} + 1}{2^N}$. As oracle BRO is executed at most $(q_{BRO} + 2q_{sign} + 1)$ times, the overall probability can be bounded by $(q_{BRO} + 2q_{sign} + 1) \cdot \dfrac{q_{BRO} + q_{BRO^{-1}} + 3q_{sign} + 1}{2^N}$.

Similarly, consider the BRO^{-1} and $Sign$ oracles. We get the following:

$$\begin{aligned}
&\Pr\big[\textbf{abort} \text{ in line 3 at the } BRO^{-1} \text{ oracle}\big] \leqslant \\
&\qquad \leqslant q_{BRO^{-1}} \cdot \frac{q_{BRO} + q_{BRO^{-1}} + 3q_{sign} + 1}{2^N}; \\
&\Pr\big[\textbf{abort} \text{ in line 19 in } \mathsf{ExecSign}^1 \text{ at the } Sign \text{ oracle}\big] = \\
&= \Pr\big[\textbf{abort} \text{ in line 24 in } \mathsf{ExecSign}^2 \text{ at the } Sign \text{ oracle}\big] \leqslant \\
&\qquad \leqslant q_{sign} \cdot \frac{q_{BRO} + q_{BRO^{-1}} + 3q_{sign} + 1}{2^N}.
\end{aligned}$$

Thus,

$$\Pr\big[\mathbf{Exp}^0(\mathcal{A}) \to 1\big] - \Pr\big[\mathbf{Exp}^1(\mathcal{A}) \to 1\big] \leqslant \frac{(q_{BRO} + q_{BRO^{-1}} + 3q_{sign} + 1)^2}{2^N}.$$

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

1 : $(\Pi^O, \Pi^S) \leftarrow (\emptyset, \emptyset), \Pi \leftarrow \Pi^O \cup \Pi^S$
2 : $\Pi_R \leftarrow \emptyset$
3 : $\Pi_Q \leftarrow \emptyset$
4 : $l \leftarrow 0,\ SESS \leftarrow \emptyset$
5 : $sid \leftarrow -1$
6 : $round_{kg} \leftarrow 0,\ ctx_{kg} \leftarrow \emptyset$
7 : $p \leftarrow \mathcal{A}()$
8 : **if** $(p \neq 1 \wedge p \neq 2)$: **return** $\perp$
9 : $(Q, d_p) \leftarrow (\varepsilon, \varepsilon)$
10 : $\{(m_i, \langle r_i, s_i \rangle)\}_{i=1}^{l+1} \xleftarrow{s} \mathcal{A}^{KGen, NewSign, Sign, BRO, BRO^{-1}, rRO, qRO}(p)$
11 : **return** $((\forall i \neq j \in \{1, \ldots, l+1\}$:
12 : $(m_i, \langle r_i, s_i \rangle) \neq (m_j, \langle r_j, s_j \rangle)) \wedge$
13 : $\wedge\ (\forall i \in \{1, \ldots, l+1\} : \mathsf{Verify}(Q, m_i, \langle r_i, s_i \rangle)))$

Oracle $BRO(\alpha)$

1 : **if** $(\alpha, \cdot) \in \Pi$: **return** $\Pi(\alpha)$
2 : $\beta \xleftarrow{\mathcal{U}} \{0, \ldots, 2^N - 1\}$
3 : **if** $((\cdot, \beta) \in \Pi)$: **abort**
4 : $\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
5 : $\Pi \leftarrow \Pi^O \cup \Pi^S$
6 : **return** β

Oracle $BRO^{-1}(\beta)$

1 : **if** $(\cdot, \beta) \in \Pi$: **return** $\Pi^{-1}(\beta)$
2 : $\alpha \xleftarrow{\mathcal{U}} \{0,1\}^N$
3 : **if** $((\alpha, \cdot) \in \Pi)$: **abort**
4 : $\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
5 : $\Pi \leftarrow \Pi^O \cup \Pi^S$
6 : **return** β

$rRO(op_R, R)$

1 : **if** $((op_R, R, \cdot) \in \Pi_R)$: **return** $\Pi_R(op_R, R)$
2 : $comm_R \xleftarrow{\mathcal{U}} \{0,1\}^n$
3 : $\Pi_R \leftarrow \Pi_R \cup \{(op_R, R, comm_R)\}$
4 : **return** $comm_R$

$qRO(op_Q, Q)$

1 : **if** $((op_Q, Q, \cdot) \in \Pi_Q)$: **return** $\Pi_Q(op_Q, Q)$
2 : $comm_Q \xleftarrow{\mathcal{U}} \{0,1\}^n$
3 : $\Pi_Q \leftarrow \Pi_Q \cup \{(op_Q, Q, comm_Q)\}$
4 : **return** $comm_Q$

$\mathsf{ExecSign}^1(state, msg)$ $(\mathbf{Exp}^1)$

1 : $round \leftarrow state.ctx.round$
2 : **if** $(round = 0)$:
3 : $e \leftarrow \mathrm{H}(state.m)$
4 : **if** $(e = 0) : e \leftarrow 1$
5 : $k_1 \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
6 : $R_1 \leftarrow k_1 P$
7 : $op_R \xleftarrow{\mathcal{U}} \{0,1\}^\kappa$
8 : $comm_R \leftarrow rRO(op_R, R_1)$
9 : $msg' \leftarrow \{comm_R\}$
10 : **else if** $(round = 1)$:
11 : $R_2 \leftarrow msg$
12 : **if** $(R_2 = -R_1)$:
13 : **return** $(state, \perp)$
14 : $R \leftarrow R_1 + R_2$
15 : **if** $((\phi(R), \cdot) \in \Pi)$:
16 : $r \leftarrow \psi(\Pi(\phi(R)))$
17 : **else**
18 : $\beta \xleftarrow{\mathcal{U}} \{0, \ldots, 2^N - 1\}$
19 : **if** $((\cdot, \beta) \in \Pi)$: **abort**
20 : $\Pi^S \leftarrow \Pi^S \cup \{(\phi(R), \beta)\}$
21 : $\Pi \leftarrow \Pi^S \cup \Pi^O$
22 : $r \leftarrow \psi(\beta)$
23 : **if** $(r = 0)$: **return** $(state, \perp)$
24 : $s_1 \leftarrow k_1 \cdot e + d_1 \cdot r$
25 : $state.flag \leftarrow 1$
26 : $msg' \leftarrow \{op_R, R_1, s_1\}$
27 : **else if** $(round = 2)$:
28 : $s_2 \leftarrow msg$
29 : $s \leftarrow s_1 + s_2$
30 : $msg' \leftarrow \varepsilon$
31 : **if** $(\mathsf{Verify}(Q, state.m, \langle r, s \rangle) = 0)$:
32 : **return** $(state, \perp)$
33 : **else**
34 : $msg' \leftarrow \varepsilon$
35 : // Update the $state.ctx$ value
36 : **return** $(state, msg')$

$\mathsf{ExecSign}^2(state, msg)$ $(\mathbf{Exp}^1)$

1 : $round \leftarrow state.ctx.round$
2 : **if** $(round = 0)$:
3 : $e \leftarrow \mathrm{H}(state.m)$
4 : **if** $(e = 0) : e \leftarrow 1$
5 : $comm_R \leftarrow msg$
6 : $k_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
7 : $R_2 \leftarrow k_2 P$
8 : $msg' \leftarrow \{R_2\}$
9 : **else if** $(round = 1)$:
10 : $(op_R, R_1, s_1) \leftarrow msg$
11 : **if** $(comm_R \neq rRO(op_R, R_1))$:
12 : **return** $(state, \perp)$
13 : **if** $(R_1 = -R_2)$:
14 : **return** $(state, \perp)$
15 : $R \leftarrow R_1 + R_2$
16 : **if** $((\phi(R), \cdot) \in \Pi)$:
17 : $r \leftarrow \psi(\Pi(\phi(R)))$
18 : **else**
19 : $\beta \xleftarrow{\mathcal{U}} \{0, \ldots, 2^N - 1\}$
20 : **if** $((\cdot, \beta) \in \Pi)$: **abort**
21 : $\Pi^S \leftarrow \Pi^S \cup \{(\phi(R), \beta)\}$
22 : $\Pi \leftarrow \Pi^S \cup \Pi^O$
23 : $r \leftarrow \psi(\beta)$
24 : $s_2 \leftarrow k_2 \cdot e + d_2 \cdot r$
25 : $s \leftarrow s_1 + s_2$
26 : $state.flag \leftarrow 1$
27 : $msg' \leftarrow \{s_2\}$
28 : **if** $(\mathsf{Verify}(Q, state.m, \langle r, s \rangle) = 0)$:
29 : **return** $(state, \perp)$
30 : **else** :
31 : $msg' \leftarrow \varepsilon$
32 : // Update the $state.ctx$ value
33 : **return** $(state, msg')$

Figure 9: $\mathbf{Exp}^1(\mathcal{A})$ for the 2p-GOST scheme in the sOMUF-PCA 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 10).

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 12.

Let construct an adversary $\mathcal{C}$ that breaks the signum-relative collision resistant property of H. 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 Π, Π_R, Π_Q implementations with lazy sampling. $\mathcal{A}$ delivers $(l+1)$ forgeries to $\mathcal{C}$, and $\mathcal{C}$ finds the signum-relative collision iff the condition in lines 11-12 is met.

Thus, we obtain the following bound:

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

The adversary $\mathcal{C}$ implements $\mathbf{Exp}^2$ and thus processes at most q_R queries to the oracle rRO, at most q_Q queries to the oracle qRO, at most $q_{BRO} + 2q_{sign} + 1$ queries to the oracle BRO, at most $q_{BRO^{-1}}$ queries to the oracle BRO^{-1} and at most q_{sign} queries to the oracles $NewSign$, at most 1 query to the oracle $KGen$, checks the collision condition and verifies the forgeries obtained from $\mathcal{A}$. Adversary $\mathcal{C}$ uses at most $3T$ computational resources since it needs to simulate signing oracle (at most $q_{sign} \leqslant T$ queries) and check the forgeries ($(q_{sign} + 1)$ pairs).

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

$(\Pi^O, \Pi^S) \leftarrow (\emptyset, \emptyset), \Pi \leftarrow \Pi^O \cup \Pi^S$
$\Pi_R \leftarrow \emptyset$
$\Pi_Q \leftarrow \emptyset$
$l \leftarrow 0,\ SESS \leftarrow \emptyset$
$sid \leftarrow -1$
$round_{kg} \leftarrow 0,\ ctx_{kg} \leftarrow \emptyset$
$p \leftarrow \mathcal{A}()$
if $(p \neq 1 \wedge p \neq 2)$: **return** $\perp$
$(Q, d_p) \leftarrow (\varepsilon, \varepsilon)$
$\{(m_i, \langle r_i, s_i\rangle)\}_{i=1}^{l+1} \xleftarrow{\$} \mathcal{A}^{KGen,NewSign,Sign,BRO,BRO^{-1},rRO,qRO}(p)$
$\forall i \neq j \in \{1, \ldots, l+1\}, m_i \neq m_j$:
  if $(\mathrm{H}(m_i) = \pm\mathrm{H}(m_j))$: **abort**
return $((\forall i \neq j \in \{1, \ldots, l+1\}$:
  $(m_i, \langle r_i, s_i\rangle) \neq (m_j, \langle r_j, s_j\rangle)) \wedge$
    $\wedge\ (\forall i \in \{1, \ldots, l+1\} : \mathsf{Verify}(Q, m_i, \langle r_i, s_i\rangle)))$

Figure 10: $\mathbf{Exp}^2(\mathcal{A})$ for the 2p-GOST scheme in the sOMUF-PCA model.

There are two cases in experiment $\mathbf{Exp}^2(\mathcal{A})$, depending on which p value the adversary $\mathcal{A}$ chooses:

1. the party $\mathbf{P_2}$ is compromised, i.e. $p = 1$;
2. the party $\mathbf{P_1}$ is compromised, i.e. $p = 2$.

Thus,

$$\begin{aligned}\Pr\left[\mathbf{Exp}^2(\mathcal{A}) \to 1\right] = \Pr\left[\mathbf{Exp}^2(\mathcal{A}) \to 1 | p = 1\right] \Pr[p = 1] + \\ + \Pr\left[\mathbf{Exp}^2(\mathcal{A}) \to 1 | p = 2\right] \Pr[p = 2] \leqslant \\ \leqslant \max\left\{\Pr\left[\mathbf{Exp}^2(\mathcal{A}) \to 1 | p = 1\right], \Pr\left[\mathbf{Exp}^2(\mathcal{A}) \to 1 | p = 2\right]\right\}.\end{aligned}$$

Let's consider both of these cases separately.

The party $\mathbf{P_2}$ is compromised. Consider the $\mathbf{Exp}^2(\mathcal{A})$ under the assumption that $p = 1$. In the further experiments we change the $\mathsf{ExecKGen}^1$ and $\mathsf{ExecSign}^1$ functions behaviour only (see Figure 11).

The $\mathsf{ExecKGen}^1$ function in $\mathbf{Exp}^3$ is the modification of the $\mathsf{ExecKGen}^1$ function in $\mathbf{Exp}^2$ (same as in $\mathbf{Exp}^1$) by adding the abort condition in case of choosing op_Q that already belongs to set Π_Q (line 9). Note that on round 0 we only select $comm_Q$ uniformly without querying random oracle qRO. We fix the values Q_1 and op_Q on the round 1 and verify if op_Q belongs to set Π_Q or not. Thus, we preserve the ability of the adversary to receive $comm_Q$ and conduct an exhaustive search using the random oracle qRO.

We should estimate the probability of this event to estimate the difference between the $\mathbf{Exp}^2$ and $\mathbf{Exp}^3$. The value op_Q is uniformly distributed in a set $\{0,1\}^\kappa$ of cardinality 2^κ. In the worst case the adversary $\mathcal{A}$ has already made all queries to the qRO oracle and thus Π_Q contains at least q_Q elements. The abort condition is met if the value op_Q hits one of elements in Π_Q. We can estimate this probability as $\frac{q_Q}{2^\kappa}$. As oracle $KGen$ is executed once, the overall probability can be bounded by $\frac{q_Q}{2^\kappa}$.

Similarly, the $\mathsf{ExecSign}^1$ function in $\mathbf{Exp}^3$ is the modification of the $\mathsf{ExecSign}^1$ function in $\mathbf{Exp}^2$ (same as in $\mathbf{Exp}^1$) by adding the abort condition in case of choosing op_R that already belongs to set Π_R (line 12). Note that on round 0 we only select $comm_R$ uniformly without using random oracle rRO. We fix the values R_1 and op_R on the round 1 and verify if op_R belongs to set Π_Q or not. Thus, we preserve the ability of the adversary to receive $comm_R$ and conduct an exhaustive search using the random oracle rRO.

We should estimate the probability of this event to estimate the difference between the $\mathbf{Exp}^2$ and $\mathbf{Exp}^3$. The value op_R is uniformly distributed in a set $\{0,1\}^\kappa$ of cardinality 2^κ. In the worst case the adversary $\mathcal{A}$ has already made all queries to the rRO and $Sign$ oracles and thus Π_R contains at least $q_R + q_{sign}$ elements. The abort condition is met if the value op_R hits one of elements in Π_R. We can estimate this probability as $\frac{q_R + q_{sign}}{2^\kappa}$. As oracle $Sign$ is executed at most q_{sign} times, the overall probability can be bounded by $q_{sign} \cdot \frac{q_R + q_{sign}}{2^\kappa}$.

Thus,

$$\Pr\big[\mathbf{Exp}^2(\mathcal{A}) \to 1\big] - \Pr\big[\mathbf{Exp}^3(\mathcal{A}) \to 1\big] \leqslant \frac{q_Q}{2^\kappa} + q_{sign} \cdot \frac{q_R + q_{sign}}{2^\kappa}.$$

$\mathsf{ExecKGen}^1(msg)$ ($\mathbf{Exp}^3$)

1 : **if** $(round_{kg} = 0)$:
2 : $comm_Q \xleftarrow{\mathcal{U}} \{0,1\}^n$
3 : $msg' \leftarrow \{comm_Q\}$
4 : **else if** $(round_{kg} = 1)$:
5 : $Q_2 \leftarrow msg$
6 : $d_1 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
7 : $Q_1 \leftarrow d_1 P$
8 : $op_Q \xleftarrow{\mathcal{U}} \{0,1\}^\kappa$
9 : **if** $((op_Q, \cdot, \cdot) \in \Pi_Q)$: **abort**
10 : $\Pi_Q \leftarrow \Pi_Q \cup \{op_Q, Q_1, comm_Q\}$
11 : **if** $(Q_2 = -Q_1)$: **return** $\perp$
12 : $Q \leftarrow Q_1 + Q_2$
13 : $msg' \leftarrow \{op_Q, Q_1\}$
14 : **else** :
15 : $msg' \leftarrow \varepsilon$
16 : $round_{kg} \leftarrow round_{kg} + 1$
17 : // Update the ctx_{kg} value
18 : **return** msg'

$\mathsf{ExecSign}^1(state, msg)$ ($\mathbf{Exp}^3$)

1 : $round \leftarrow state.ctx.round$
2 : **if** $(round = 0)$:
3 : $e \leftarrow \mathrm{H}(state.m)$
4 : **if** $(e = 0) : e \leftarrow 1$
5 : $comm_R \xleftarrow{\mathcal{U}} \{0,1\}^l$
6 : $msg' \leftarrow \{comm_R\}$
7 : **else if** $(round = 1)$:
8 : $R_2 \leftarrow msg$
9 : $k_1 \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
10 : $R_1 \leftarrow k_1 P$
11 : $op_R \xleftarrow{\mathcal{U}} \{0,1\}^\kappa$
12 : **if** $((op_R, \cdot, \cdot) \in \Pi_R)$: **abort**
13 : $\Pi_R \leftarrow \Pi_R \cup \{op_R, R_1, comm_R\}$
14 : **if** $(R_2 = -R_1)$:
15 : **return** $(state, \perp)$
16 : $R \leftarrow R_1 + R_2$
17 : **if** $((\phi(R), \cdot) \in \Pi)$:
18 : $r \leftarrow \psi(\Pi(\phi(R)))$
19 : **else** :
20 : $\beta \xleftarrow{\mathcal{U}} \{0, \ldots, 2^N - 1\}$
21 : **if** $((\cdot, \beta) \in \Pi)$: **abort**
22 : $\Pi^S \leftarrow \Pi^S \cup \{(\phi(R), \beta)\}$
23 : $\Pi \leftarrow \Pi^S \cup \Pi^O$
24 : $r \leftarrow \psi(\beta)$
25 : **if** $(r = 0)$: **return** $(state, \perp)$
26 : $s_1 \leftarrow k_1 \cdot e + d_1 \cdot r$
27 : $state.flag \leftarrow 1$
28 : $msg' \leftarrow \{op_R, R_1, s_1\}$
29 : **else if** $(round = 2)$:
30 : $s_2 \leftarrow msg$
31 : $s \leftarrow s_1 + s_2$
32 : $msg' \leftarrow \varepsilon$
33 : **if** $(\mathsf{Verify}(Q, state.m, \langle r, s \rangle) = 0)$:
34 : **return** $(state, \perp)$
35 : **else**
36 : $msg' \leftarrow \varepsilon$
37 : // Update the $state.ctx$ value
38 : **return** $(state, msg')$

$\mathsf{ExecSign}^1(state, msg)$ ($\mathbf{Exp}^4$)

1 : $round \leftarrow state.ctx.round$
2 : **if** $(round = 0)$:
3 : $e \leftarrow \mathrm{H}(state.m)$
4 : **if** $(e = 0) : e \leftarrow 1$
5 : $comm_R \xleftarrow{\mathcal{U}} \{0,1\}^l$
6 : $msg' \leftarrow \{comm_R\}$
7 : **else if** $(round = 1)$:
8 : $R_2 \leftarrow msg$
9 : $\beta \xleftarrow{\mathcal{U}} \{0, \ldots, 2^N - 1\}$
10 : $r \leftarrow \psi(\beta)$
11 : $s_1 \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
12 : $R_1 \leftarrow e^{-1} s_1 P - e^{-1} r Q_1$
13 : $op_R \xleftarrow{\mathcal{U}} \{0,1\}^\kappa$
14 : **if** $((op_R, \cdot, \cdot) \in \Pi_R)$: **abort**
15 : $\Pi_R \leftarrow \Pi_R \cup \{op_R, R_1, comm_R\}$
16 : **if** $(R_2 = -R_1)$:
17 : **return** $(state, \perp)$
18 : $R \leftarrow R_1 + R_2$
19 : **if** $(r = 0)$: **return** $(state, \perp)$
20 : **if** $((\phi(R), \cdot) \in \Pi)$: **abort**
21 : **if** $((\cdot, \beta) \in \Pi)$: **abort**
22 : $\Pi^S \leftarrow \Pi^S \cup \{(\phi(R), \beta)\}$
23 : $\Pi \leftarrow \Pi^S \cup \Pi^O$
24 : $state.flag \leftarrow 1$
25 : $msg' \leftarrow\leftarrow \{op_R, R_1, s_1\}$
26 : **else if** $(round = 2)$:
27 : $s_2 \leftarrow msg$
28 : $s \leftarrow s_1 + s_2$
29 : $msg' \leftarrow \varepsilon$
30 : **if** $(\mathsf{Verify}(Q, state.m, \langle r, s \rangle) = 0)$:
31 : **return** $(state, \perp)$
32 : **else**
33 : $msg' \leftarrow \varepsilon$
34 : // Update the $state.ctx$ value
35 : **return** $(state', msg')$

Figure 11: The $\mathsf{ExecKGen}^1$ and $\mathsf{ExecSign}^1$ functions in $\mathbf{Exp}^3(\mathcal{A})$, $\mathbf{Exp}^4(\mathcal{A})$

The signing oracle in the $\mathbf{Exp}^4$ gets along with only public information. Values β and s_1 are randomly chosen from the relevant sets and then point R_1 is constructed. We define the corresponding pair in Π implementation by saving this pair in the Π^S set. Note that if we couldn't do so (i.e., β already belongs to the Π), the abort condition is met like in the $\mathbf{Exp}^3$.

Consider the distribution on R_1 and s_1, that are returned by the *Sign* oracle on the round 1. In the $\mathbf{Exp}^3$ value k_1 is distributed uniformly on $\mathbb{Z}_q$, thus R_1 is uniformly distributed. The value r is independent on k_1 (due to bijective random oracle) and thus s_1 value is also uniformly distributed on $\mathbb{Z}_q$.

In the $\mathbf{Exp}^4$ values R_1 and s_1 are also distributed uniformly on the corresponding sets except of the values that lead to $\phi(R)$ that already belongs to Π. Let's estimate the probability of these «bad» event. The values s_1 and r are uniformly distributed on a set $\mathbb{Z}_q$ and are chosen independently. Then the value R_1 (and thus R) is uniformly distributed on

a set of cardinality q. In the worst case the adversary $\mathcal{A}$ has already made all queries to the BRO, BRO^{-1}, $Sign$ oracles and thus Π contains at least $(q_{BRO}+q_{BRO^{-1}}+2q_{sign})$ elements. Note that here we do not take into account the queries to the BRO oracle made during finalizing the experiment (verifying the forgeries), since they are made after all queries to the $Sign$ oracle. The abort condition is met if the value $\phi(R)$ hits one of elements in Π. We can estimate this probability as $\dfrac{q_{BRO}+q_{BRO^{-1}}+2q_{sign}}{q}$. As oracle $Sign$ is executed at most q_{sign} times, the overall probability can be bounded by $q_{sign}\cdot\dfrac{q_{BRO}+q_{BRO^{-1}}+2q_{sign}}{q}$.

Thus, we conclude that

$$\Pr\big[\mathbf{Exp}^3(\mathcal{A})\to 1\big]-\Pr\big[\mathbf{Exp}^4(\mathcal{A})\to 1\big]\leqslant q_{sign}\cdot\frac{q_{BRO}+q_{BRO^{-1}}+2q_{sign}}{q}.$$

Let construct the adversary $\mathcal{B}$ for the GOST scheme in the sUF-KO model that uses $\mathcal{A}$ as the black box (see Figure 12).

$\mathcal{B}^{BRO^*,BRO^{*-1}}(Q,\mathcal{A})$

1 : $(\Pi^O,\Pi^S) \leftarrow (\emptyset,\emptyset), \Pi \leftarrow \Pi^O \cup \Pi^S$
2 : $\Pi_R \leftarrow \emptyset$
3 : $\Pi_Q \leftarrow \emptyset$
4 : $l \leftarrow 0,\ SESS \leftarrow \emptyset$
5 : $sid \leftarrow -1$
6 : $round_{kg} \leftarrow 0,\ ctx_{kg} \leftarrow \emptyset$
7 : $p \leftarrow \mathcal{A}()$
8 : **if** $(p \neq 1 \wedge p \neq 2)$: **return** $\perp$
9 : $\{(m_i, \langle r_i, s_i\rangle)\}_{i=1}^{l+1} \xleftarrow{\$} \mathcal{A}^{KGen,NewSign,Sign,BRO,BRO^{-1},rRO,qRO}(p)$
10 : $\forall i \neq j \in \{1,\ldots,l+1\}, m_i \neq m_j$:
11 : **if** $(\mathrm{H}(m_i) = \pm\mathrm{H}(m_j))$: **abort**
12 : **if** $((\exists i \neq j \in \{1,\ldots,l+1\} : (m_i, \langle r_i, s_i\rangle) = (m_j, \langle r_j, s_j\rangle))$:
13 : **abort**
14 : **for** $i \in \{1,\ldots,l+1\}$:
15 : $e_i \leftarrow \mathrm{H}(m_i)$
16 : **if** $(e_i = 0) : e_i \leftarrow 1$
17 : $R_i \leftarrow e_i^{-1}(s_iP - r_iQ)$
18 : **if** $\psi(SimBRO(\phi(R_i))) \neq r_i$: **abort**
19 : **if** $(\phi(R_i), \cdot) \in \Pi^O$: **return** $(m_i, \langle r_i, s_i\rangle)$
20 : Find $i,j : ((\phi(R_i), \cdot) \in \Pi^S) \wedge (\phi(R_i) = \phi(R_j))$
21 : Compute d
22 : $(m, \langle r, s\rangle) \xleftarrow{\$} \mathsf{GOST.Sign}(d, m)$
23 : **return** $(m, \langle r, s\rangle)$

Oracle $SimBRO(\alpha)$

1 : **if** $(\alpha, \cdot) \in \Pi$: **return** $\Pi(\alpha)$
2 : $\beta \leftarrow BRO^*(\alpha)$
3 : **if** $((\cdot\beta) \in \Pi)$: **abort**
4 : $\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
5 : $\Pi \leftarrow \Pi^O \cup \Pi^S$
6 : **return** β

Oracle $SimBRO^{-1}(\beta)$

1 : **if** $(\cdot, \beta) \in \Pi$: **return** $\Pi^{-1}(\beta)$
2 : $\alpha \leftarrow BRO^{*-1}(\beta)$
3 : **if** $((\alpha, \cdot) \in \Pi)$: **abort**
4 : $\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
5 : $\Pi \leftarrow \Pi^O \cup \Pi^S$
6 : **return** β

$\mathsf{ExecKGen}^1(msg)$

1 : **if** $(round_{kg} = 0)$:
2 : $comm_R \xleftarrow{\mathcal{U}} \{0,1\}^n$
3 : $msg' \leftarrow \{comm_Q\}$
4 : **else if** $(round_{kg} = 1)$:
5 : $Q_2 \leftarrow msg$
6 : $Q_1 \leftarrow Q - Q_2$
7 : $op_Q \xleftarrow{\mathcal{U}} \{0,1\}^\kappa$
8 : **if** $((op_Q, \cdot, \cdot) \in \Pi_Q)$: **abort**
9 : $\Pi_Q \leftarrow \Pi_Q \cup \{op_Q, Q_1, comm_Q\}$
10 : **if** $(Q_2 = -Q_1)$: **return** $\perp$
11 : $Q \leftarrow Q_1 + Q_2$
12 : $msg' \leftarrow \{op_Q, Q_1\}$
13 : **else** :
14 : $msg' \leftarrow \varepsilon$
15 : $round_{kg} \leftarrow round_{kg} + 1$
16 : // Update the ctx_{kg} value
17 : **return** msg'

Figure 12: The adversary $\mathcal{B}$ for the GOST scheme in the sUF-KO model that uses $\mathcal{A}$ as the black box

Adversary $\mathcal{B}$ simulates the rRO, qRO, $NewSign$ and $Sign$ oracles to answer the $\mathcal{A}$ queries as the corresponding oracles in the $\mathbf{Exp}^4$. Adversary $\mathcal{B}$ simulates the BRO, BRO^{-1} oracles by translating the queries to its own oracle (see $SimBRO$ and $SimBRO^{-1}$). Adversary $\mathcal{B}$ simulates $KGen$ oracle similar to the oracle $KGen$ in the $\mathbf{Exp}^4$ with the modification of the $\mathsf{ExecKGen}^1$ function. Adversary $\mathcal{B}$ sets Q_1 value (after receiving Q_2) in such a way that the resulting public key iis equal to the public key Q, provided by its challenger.

After receiving $l+1$ forgeries from $\mathcal{A}$, $\mathcal{B}$ finds the suitable forgery relative to its own challenger. Assume that $\mathcal{A}$ delivers valid pairs $\{(m_i, \langle r_i, s_i\rangle)\}_{i=1}^{l+1}$. This means that the set Π contains all corresponding pairs $(\phi(R_i), \beta), i = 1,\ldots,l+1$: either these pairs were already in the Π before verification check in line 18 or were saved after $SimBRO$ call during

this check. There are two possible cases. If there exists at least one pair $(\phi(R_i), \beta) \in \Pi^O$, then it is already a valid forgery with respect to the oracles BRO^*, BRO^{*-1} and $\mathcal{B}$ can simply forward it to its own challenger. If all pairs $(\phi(R_i), \beta) \in \Pi^S, i = 1, \ldots, l+1$, $\mathcal{B}$ can recover the signing key d as described below and construct the new forgery for an arbitrary message.

Note that there are at least l pairs in Π^S, because adding a pair to the set Π^S is performed only during the *Sign* oracle execution simultaneously with incrementing the counter l of successful sessions. Thus, if pairs $(\phi(R_i), \beta) \in \Pi^S, i = 1, \ldots, l+1$, then there are two of them with indexes i, j such as $\phi(R_i) = \phi(R_j)$. This means that $r_i = r_j = \psi(\beta) = r$ in the corresponding forgeries. The adversary $\mathcal{B}$ knows the corresponding $e_i = \mathrm{H}(m_i)$, $e_j = \mathrm{H}(m_j)$.

The equations $\phi(R_i) = \phi(R_j)$ implies $R_i = \pm R_j$ and thus $k_i = \pm k_j = k$. So the following linear equation system holds:

$$\begin{cases} s_i &= ke_i + dr; \\ s_j &= \pm ke_j + dr; \end{cases}$$

There are two unknown variables k and d in the system above. This system has a unique solution whenever $e_i \neq \pm e_j$. Observe that case $e_i = \pm e_j$ and thus $\mathrm{H}(m_i) = \pm\mathrm{H}(m_j)$ is excluded by lines 10, 11, if $m_i \neq m_j$. The $m_i = m_j$ condition (together with $r_i = r_j$ condition) implies either $(m_i, \langle r_i, s_i\rangle) = (m_j, \langle r_j, s_j\rangle)$, that is excluded by lines 12, 13, or $k_i = -k_j$, that still allows to compute d from the system equation. Summing all, we can always compute d if all pairs $(\phi(R_i), \beta), i = 1, \ldots, l+1$, belongs to Π^S.

We conclude that if $\mathcal{A}$ delivers valid $l+1$ forgeries, $\mathcal{B}$ delivers a valid forgery to its own challenger and

$$\Pr\big[\mathbf{Exp}^4(\mathcal{A}) \to 1\big] = \Pr\big[\mathbf{Exp}^{\mathsf{sUF\text{-}KO}}_{\mathsf{GOST}}(\mathcal{B}) \to 1\big].$$

Note that the number of queries made by $\mathcal{B}$ to the BRO^* and BRO^{*-1} oracles is at most $q_{BRO} + 2q_{sign} + 1$ and $q_{BRO^{-1}}$ respectively. The adversary $\mathcal{B}$ needs the same amount of computational resources as $\mathcal{C}$.

Thus, we summarize the obtained bounds in case the party P_2 is compromised:

$$\begin{aligned}\Pr\big[\mathbf{Exp}^2(\mathcal{A}) \to 1\big] &= \big(\Pr\big[\mathbf{Exp}^2(\mathcal{A}) \to 1\big] - \Pr\big[\mathbf{Exp}^3(\mathcal{A}) \to 1\big]\big) + \\ &+ \big(\Pr\big[\mathbf{Exp}^3(\mathcal{A}) \to 1\big] - \Pr\big[\mathbf{Exp}^4(\mathcal{A}) \to 1\big]\big) + \Pr\big[\mathbf{Exp}^4(\mathcal{A}) \to 1\big] \leqslant \\ &\leqslant \frac{q_Q}{2^\kappa} + q_{sign} \cdot \frac{q_R + q_{sign}}{2^\kappa} + q_{sign} \cdot \frac{q_{BRO} + q_{BRO^{-1}} + 2q_{sign}}{q} + \\ &+ \Pr\big[\mathbf{Exp}^{\mathsf{sUF\text{-}KO}}_{\mathsf{GOST}}(\mathcal{B}) \to 1\big] = \frac{q_Q}{2^\kappa} + q_{sign} \cdot \frac{q_R + q_{sign}}{2^\kappa} + \\ &+ q_{sign} \cdot \frac{q_{BRO} + q_{BRO^{-1}} + 2q_{sign}}{q} + \mathsf{Adv}^{\mathsf{sUF\text{-}KO}}_{\mathsf{GOST}}(\mathcal{B}).\end{aligned}$$

The party $\mathbf{P_1}$ is compromised. Consider the $\mathbf{Exp}^2$ under the assumption that $p = 2$. In the further experiments we change the $\mathsf{ExecKGen}^2$ and $\mathsf{ExecSign}^2$ functions behaviour only (see Figure 13).

The $\mathsf{ExecKGen}^2$ function in $\mathbf{Exp}^3$ is the modification of the $\mathsf{ExecKGen}^2$ function in $\mathbf{Exp}^2$ (same as in $\mathbf{Exp}^1$) by adding the abort condition in case of receiving $comm_Q$ that does not belong to set Π_Q (lines 4, 5, 12). Note that on round 0 we only set $flag_Q$ and abort on the round 1. Thus, we preserve the ability of the adversary to receive Q_2.

We should estimate the probability of this event to estimate the difference between the $\mathbf{Exp}^2$ and $\mathbf{Exp}^3$. The value $comm_Q$ belongs to the set $\{0,1\}^n$ of cardinality 2^n. In the worst case the adversary $\mathcal{A}$ has already made all queries to the qRO oracle and thus Π_Q contains at least q_Q elements. The abort condition is met if the value $comm_Q$ hits one of elements in Π_Q. We can estimate this probability as $\frac{q_Q}{2^n}$. As oracle $KGen$ is executed only once, the overall probability can be bounded by $\frac{q_Q}{2^n}$.

Similarly, the $\mathsf{ExecSign}^2$ function in $\mathbf{Exp}^3$ is the modification of the $\mathsf{ExecSign}^2$ function in $\mathbf{Exp}^2$ (same as in $\mathbf{Exp}^1$) by adding the abort condition in case of receiving $comm_R$ that does not belong to set Π_R (lines 7, 8, 16). Note that on round 0 we only set $flag_R$ and abort on the round 1. Thus, we preserve the ability of the adversary to receive R_2.

We should estimate the probability of this event to estimate the difference between the $\mathbf{Exp}^2$ and $\mathbf{Exp}^3$. The value $comm_R$ belongs to the set $\{0,1\}^n$ of cardinality 2^n. In the worst case the adversary $\mathcal{A}$ has already made all queries to the rRO and $Sign$ oracles and thus Π_R contains at least $q_R + q_{sign}$ elements. The abort condition is met if the value $comm_R$ hits one of elements in Π_R. We can estimate this probability as $\frac{q_R + q_{sign}}{2^n}$. As oracle $Sign$ is executed at most q_{sign} times, the overall probability can be bounded by $q_{sign} \cdot \frac{q_R + q_{sign}}{2^n}$.

Thus,

$$\Pr\left[\mathbf{Exp}^2(\mathcal{A}) \to 1\right] - \Pr\left[\mathbf{Exp}^3(\mathcal{A}) \to 1\right] \leqslant \frac{q_Q}{2^n} + q_{sign} \cdot \frac{q_R + q_{sign}}{2^n}.$$

$\mathsf{ExecKGen}^2(msg)$ ($\mathbf{Exp}^3$)

1 : **if** $(round_{kg} = 0)$:
2 : $flag_Q \leftarrow 0$
3 : $comm_Q \leftarrow msg$
4 : **if** $((\cdot, \cdot, comm_Q) \notin \Pi_Q)$:
5 : $flag_Q \leftarrow 1$
6 : $d_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q^*$
7 : $Q_2 \leftarrow d_2 P$
8 : $msg' \leftarrow \{Q_2\}$
9 : **else if** $(round_{kg} = 1)$:
10 : $op_Q, Q_1 \leftarrow msg$
11 : **if** $(comm_Q \neq qRO(op_Q, Q_1))$: **return** $\perp$
12 : **if** $flag_Q$: **abort**
13 : **if** $(Q_2 = -Q_1)$: **return** $\perp$
14 : $Q \leftarrow Q_1 + Q_2$
15 : $msg' \leftarrow \varepsilon$
16 : **else** :
17 : $msg' \leftarrow \varepsilon$
18 : $round_{kg} \leftarrow round_{kg} + 1$
19 : // Update the ctx_{kg} value
20 : **return** msg'

$\mathsf{ExecSign}^2(state, msg)$ ($\mathbf{Exp}^3$)

1 : $round \leftarrow state.ctx.round$
2 : **if** $(round = 0)$:
3 : $flag_R \leftarrow 0$
4 : $e \leftarrow \mathrm{H}(state.m)$
5 : **if** $(e = 0) : e \leftarrow 1$
6 : $comm_R \leftarrow msg$
7 : **if** $((\cdot, \cdot, comm_R) \notin \Pi_R)$:
8 : $flag_R \leftarrow 1$
9 : $k_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
10 : $R_2 \leftarrow k_2 P$
11 : $msg' \leftarrow \{R_2\}$
12 : **else if** $(round = 1)$:
13 : $(op_R, R_1, s_1) \leftarrow msg$
14 : **if** $(comm_R \neq rRO(op_R, R_1))$:
15 : **return** $(state, \perp)$
16 : **if** $flag_R$: **abort**
17 : **if** $(R_1 = -R_2)$:
18 : **return** $(state, \perp)$
19 : $R \leftarrow R_1 + R_2$
20 : **if** $((\phi(R), \cdot) \in \Pi)$:
21 : $r \leftarrow \psi(\Pi(\phi(R)))$
22 : **else**
23 : $\beta \xleftarrow{\mathcal{U}} \{0, \ldots, 2^N - 1\}$
24 : **if** $((\cdot, \beta) \in \Pi)$: **abort**
25 : $\Pi^S \leftarrow \Pi^S \cup \{(\phi(R), \beta)\}$
26 : $\Pi \leftarrow \Pi^S \cup \Pi^O$
27 : $r \leftarrow \psi(\beta)$
28 : $s_2 \leftarrow k_2 \cdot e + d_2 \cdot r$
29 : $s \leftarrow s_1 + s_2$
30 : $state.flag \leftarrow 1$
31 : $msg' \leftarrow \{s_2\}$
32 : **if** $(\mathsf{Verify}(Q, state.m, \langle r, s \rangle) = 0)$:
33 : **return** $(state, \perp)$
34 : **else** :
35 : $msg' \leftarrow \varepsilon$
36 : // Update the $state.ctx$ value
37 : **return** $(state, msg')$

$\mathsf{ExecSign}^2(state, msg)$ ($\mathbf{Exp}^4$)

1 : $round \leftarrow state.ctx.round$
2 : **if** $(round = 0)$:
3 : $flag_R \leftarrow 0$
4 : $e \leftarrow \mathrm{H}(state.m)$
5 : **if** $(e = 0) : e \leftarrow 1$
6 : $comm_R \leftarrow msg$
7 : **if** $((\cdot, \cdot, comm_R) \notin \Pi_R)$:
8 : $flag_R \leftarrow 1$
9 : $\beta \xleftarrow{\mathcal{U}} \{0, \ldots, 2^N - 1\}$
10 : $r \leftarrow \psi(\beta)$
11 : $s_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
12 : $R_2 \leftarrow e^{-1} s_2 P - e^{-1} r Q_2$
13 : $msg' \leftarrow \{R_2\}$
14 : **else if** $(round = 1)$:
15 : $(op_1, R_1, s_1) \leftarrow msg$
16 : **if** $(comm_R \neq rRO(op_R, R_1))$:
17 : **return** $(state, \perp)$
18 : **if** $flag_R$: **abort**
19 : **if** $(R_1 = -R_2)$:
20 : **return** $(state, \perp)$
21 : $R \leftarrow R_1 + R_2$
22 : **if** $((\phi(R), \cdot) \in \Pi)$: **abort**
23 : **if** $((\cdot, \beta) \in \Pi)$: **abort**
24 : $\Pi^S \leftarrow \Pi^S \cup \{(\phi(R), \beta)\}$
25 : $\Pi \leftarrow \Pi^S \cup \Pi^O$
26 : $s \leftarrow s_1 + s_2$
27 : $state.flag \leftarrow 1$
28 : $msg' \leftarrow \{s_2\}$
29 : **if** $(\mathsf{Verify}(Q, state.m, \langle r, s \rangle) = 0)$:
30 : **return** $(state, \perp)$
31 : **else** :
32 : $msg' \leftarrow \varepsilon$
33 : // Update the $state.ctx$ value
34 : **return** $(state, msg')$

Figure 13: The $\mathsf{ExecKGen}^2$ and $\mathsf{ExecSign}^2$ functions in $\mathbf{Exp}^3(\mathcal{A})$, $\mathbf{Exp}^4(\mathcal{A})$

The signature oracle in the $\mathbf{Exp}^4$ gets along with only public information. Values β and s_2 are randomly chosen from the relevant sets and then point R_2 is constructed. We define the corresponding pair in Π implementation by saving this pair in the Π^S set. Note that if we couldn't do so (i.e., β already belongs to the Π), the abort condition is met like in the $\mathbf{Exp}^3$.

Consider the distribution on R_2 and s_2, that are returned by the *Sign* oracle on the rounds 1 and 2 respectively. In the $\mathbf{Exp}^3$ value k_2 is distributed uniformly on $\mathbb{Z}_q$, thus R_2 is uniformly distributed. The value r is independent on k_2 (due to bijective random oracle) and thus s_2 value is also uniformly distributed on $\mathbb{Z}_q$.

In the $\mathbf{Exp}^4$ values R_2 and s_2 are also distributed uniformly on the corresponding sets except of the values that lead to $\phi(R)$ that already belongs to Π. Let's estimate the

probability of these «bad» event. The values s_2 and r are uniformly distributed on a set $\mathbb{Z}_q$ and are chosen independently. Then the value R_2 (and thus R) is uniformly distributed on a set of cardinality q. In the worst case the adversary $\mathcal{A}$ has already made all queries to the BRO, BRO^{-1}, $Sign$ oracles and thus Π contains at least $(q_{BRO}+q_{BRO^{-1}}+2q_{sign})$ elements. Note that here we do not take into account the queries to the BRO oracle made during finalizing the experiment (verifying the forgeries), since they are made after all queries to the $Sign$ oracle. The abort condition is met if the value $\phi(R)$ hits one of elements in Π. We can estimate this probability as $\dfrac{q_{BRO}+q_{BRO^{-1}}+2q_{sign}}{q}$. As oracle $Sign$ is executed at most q_{sign} times, the overall probability can be bounded by $q_{sign}\cdot\dfrac{q_{BRO}+q_{BRO^{-1}}+2q_{sign}}{q}$.

Thus, we conclude that

$$\Pr\left[\mathbf{Exp}^3(\mathcal{A})\to 1\right]-\Pr\left[\mathbf{Exp}^4(\mathcal{A})\to 1\right]\leqslant q_{sign}\cdot\frac{q_{BRO}+q_{BRO^{-1}}+2q_{sign}}{q}.$$

Let construct the adversary $\mathcal{B}$ for the GOST scheme in the sUF-KO model that uses $\mathcal{A}$ as the black box (see Figure 14).

$\mathcal{B}^{BRO^*,BRO^{*-1}}(Q,\mathcal{A})$

$(\Pi^O,\Pi^S) \leftarrow (\emptyset,\emptyset), \Pi \leftarrow \Pi^O \cup \Pi^S$
$\Pi_R \leftarrow \emptyset$
$\Pi_Q \leftarrow \emptyset$
$l \leftarrow 0,\ SESS \leftarrow \emptyset$
$sid \leftarrow -1$
$round_{kg} \leftarrow 0,\ ctx_{kg} \leftarrow \emptyset$
$p \leftarrow \mathcal{A}()$
if $(p \neq 1 \wedge p \neq 2)$: **return** $\perp$
$\{(m_i, \langle r_i, s_i\rangle)\}_{i=1}^{l+1} \xleftarrow{\$} \mathcal{A}^{KGen,NewSign,Sign,BRO,BRO^{-1},rRO,qRO}(p)$
$\forall i \neq j \in \{1,\ldots,l+1\}, m_i \neq m_j$:
  if $(\mathrm{H}(m_i) = \pm\mathrm{H}(m_j))$: **abort**
if $((\exists i \neq j \in \{1,\ldots,l+1\} : (m_i, \langle r_i, s_i\rangle) = (m_j, \langle r_j, s_j\rangle))$:
  abort
for $i \in \{1,\ldots,l+1\}$:
  $e_i \leftarrow \mathrm{H}(m_i)$
  if $(e_i = 0) : e_i \leftarrow 1$
  $R_i \leftarrow e_i^{-1}(s_iP - r_iQ)$
  if $\psi(SimBRO(\phi(R_i))) \neq r_i$: **abort**
  if $(\phi(R_i), \cdot) \in \Pi^O$: **return** $(m_i, \langle r_i, s_i\rangle)$
Find $i,j : ((\phi(R_i), \cdot) \in \Pi^S) \wedge (\phi(R_i) = \phi(R_j))$
Compute d
$(m, \langle r, s\rangle) \xleftarrow{\$} \mathsf{GOST.Sign}(d, m)$
return $(m, \langle r, s\rangle)$

Oracle $SimBRO(\alpha)$

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

Oracle $SimBRO^{-1}(\beta)$

if $(\cdot, \beta) \in \Pi$: **return** $\Pi^{-1}(\beta)$
$\alpha \leftarrow BRO^{*-1}(\beta)$
if $((\alpha, \cdot) \in \Pi)$: **abort**
$\Pi^O \leftarrow \Pi^O \cup \{(\alpha, \beta)\}$
$\Pi \leftarrow \Pi^O \cup \Pi^S$
return β

$\mathsf{ExecKGen}^2(msg)$

if $(round_{kg} = 0)$:
  $flag_Q \leftarrow 0$
  $comm_Q \leftarrow msg$
  if $((\cdot, \cdot, comm_Q) \notin \Pi_Q)$:
    $flag_Q \leftarrow 1$
    $d_2 \xleftarrow{\mathcal{U}} \mathbb{Z}_q$
    $Q_2 \leftarrow d_2P$
else
    $Q_1 \leftarrow \Pi_Q[comm_Q]$
    $Q_2 \leftarrow Q - Q_1$
  $msg' \leftarrow \{Q_2\}$
  else if $(round_{kg} = 1)$:
  $op_Q, Q_1 \leftarrow msg$
  if $(comm_Q \neq qRO(op_Q, Q_1))$: **return** $\perp$
  if $flag_Q$: **abort**
  if $(Q_2 = -Q_1)$: **return** $\perp$
  $Q \leftarrow Q_1 + Q_2$
  $msg' \leftarrow \varepsilon$
else :
  $msg' \leftarrow \varepsilon$
$round_{kg} \leftarrow round_{kg} + 1$
  // Update the ctx_{kg} value
return msg'

Figure 14: The adversary $\mathcal{B}$ for the GOST scheme in the sUF-KO model that uses $\mathcal{A}$ as the black box

Adversary $\mathcal{B}$ simulates the rRO, qRO, $NewSign$ and $Sign$ oracles to answer the $\mathcal{A}$ queries as the corresponding oracles in the $\mathbf{Exp}^4$. Adversary $\mathcal{B}$ simulates the BRO, BRO^{-1} oracles by translating the queries to its own oracle (see $SimBRO$ and $SimBRO^{-1}$). Adversary $\mathcal{B}$ simulates $KGen$ oracle similar to the oracle $KGen$ in the $\mathbf{Exp}^4$ with the modification of the $\mathsf{ExecKGen}^1$ function. It uses the Π_Q set to know the Q_1 value from the received commitment $comm_Q$ and then sets Q_2 in such a way that the resulting public key is equal to the public key Q, provided by its challenger.

Receiving the forgeries from $\mathcal{A}$, the adversary $\mathcal{B}$ constructs the forgery for its own

challenger in the same way as defined in case when party P_2 is compromised. So, if $\mathcal{A}$ delivers a valid $l+1$ pairs, $\mathcal{B}$ delivers a valid forgery to its own challenger and

$$\Pr\big[\mathbf{Exp}^4(\mathcal{A}) \to 1\big] = \Pr\big[\mathbf{Exp}^{\mathsf{sUF\text{-}KO}}_{\mathsf{GOST}}(\mathcal{B}) \to 1\big].$$

The number of queries made by $\mathcal{B}$ to the BRO^* and BRO^{*-1} oracles is at most $q_{BRO} + 2q_{sign} + 1$ and $q_{BRO^{-1}}$ respectively. The adversary $\mathcal{B}$ needs the same amount of computational resources as $\mathcal{C}$.

Thus, we summarize the obtained bounds in case the party P_1 is compromised:

$$\begin{aligned}
\Pr\big[\mathbf{Exp}^2(\mathcal{A}) \to 1\big] &= \big(\Pr\big[\mathbf{Exp}^2(\mathcal{A}) \to 1\big] - \Pr\big[\mathbf{Exp}^3(\mathcal{A}) \to 1\big]\big) + \\
&+ \big(\Pr\big[\mathbf{Exp}^3(\mathcal{A}) \to 1\big] - \Pr\big[\mathbf{Exp}^4(\mathcal{A}) \to 1\big]\big) + \Pr\big[\mathbf{Exp}^4(\mathcal{A}) \to 1\big] \leqslant \\
&\leqslant \frac{q_Q}{2^n} + q_{sign} \cdot \frac{q_R + q_{sign}}{2^n} + q_{sign} \cdot \frac{q_{BRO} + q_{BRO^{-1}} + 2q_{sign}}{q} + \\
&+ \Pr\big[\mathbf{Exp}^{\mathsf{sUF\text{-}KO}}_{\mathsf{GOST}}(\mathcal{B}) \to 1\big] = \frac{q_Q}{2^n} + q_{sign} \cdot \frac{q_R + q_{sign}}{2^n} + \\
&+ q_{sign} \cdot \frac{q_{BRO} + q_{BRO^{-1}} + 2q_{sign}}{q} + \mathsf{Adv}^{\mathsf{sUF\text{-}KO}}_{\mathsf{GOST}}(\mathcal{B}).
\end{aligned}$$

All in all we prove:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{sOMUF\text{-}PCA}}_{\mathsf{2p\text{-}GOST}}(\mathcal{A}) &= \Pr\big[\mathbf{Exp}^0(\mathcal{A}) \to 1\big] = \\
&= \big(\Pr\big[\mathbf{Exp}^0(\mathcal{A}) \to 1\big] - \Pr\big[\mathbf{Exp}^1(\mathcal{A}) \to 1\big]\big) + \\
&+ \big(\Pr\big[\mathbf{Exp}^1(\mathcal{A}) \to 1\big] - \Pr\big[\mathbf{Exp}^2(\mathcal{A}) \to 1\big]\big) + \Pr\big[\mathbf{Exp}^2(\mathcal{A}) \to 1\big] \leq \\
&\leqslant \frac{(q_{BRO} + q_{BRO^{-1}} + 3q_{sign} + 1)^2}{2^N} + \mathsf{Adv}^{\mathsf{SCR}}_{\mathrm{H}}(\mathcal{C}) + \frac{q_Q}{2^{\min\{\kappa,n\}}} + \\
&+ q_{sign} \cdot \frac{q_R + q_{sign}}{2^{\min\{\kappa,n\}}} + q_{sign} \cdot \frac{q_{BRO} + q_{BRO^{-1}} + 2q_{sign}}{q} + \mathsf{Adv}^{\mathsf{sUF\text{-}KO}}_{\mathsf{GOST}}(\mathcal{B}) \leqslant \\
&\leqslant \mathsf{Adv}^{\mathsf{SCR}}_{\mathrm{H}}(\mathcal{C}) + \mathsf{Adv}^{\mathsf{sUF\text{-}KO}}_{\mathsf{GOST}}(\mathcal{B}) + \frac{q_Q + q_{sign} \cdot (q_R + q_{sign})}{2^{\min\{\kappa,n\}}} + \\
&+ \frac{2(q_{BRO} + q_{BRO^{-1}} + 3q_{sign} + 1)^2}{q}.
\end{aligned}$$

□