Server-Supported Decryption for Mobile Devices

Johanna Maria Kirss[1,2(✉)], Peeter Laud[1(✉)], Nikita Snetkov[1,3], and Jelizaveta Vakarjuk[1]

[1] Cybernetica AS, Tallinn, Estonia
{johanna.kirss,peeter.laud,nikita.snetkov,
jelizaveta.vakarjuk}@cyber.ee
[2] University of Tartu, Tartu, Estonia
[3] Tallinn University of Technology, Tallinn, Estonia

Abstract. We propose a threshold encryption scheme with two-party decryption, where one of the keyshares may be stored and used in a device that is able to provide only weak security for it. We state the security properties the scheme needs to have to support such use-cases, and construct a scheme with these properties. Our construction is based on the ElGamal cryptosystem, with additional zero-knowledge proofs that can provide IND-CCA security, and resistance to offline guessing attacks.

Keywords: Threshold encryption schemes · Offline guessing attacks

1 Introduction

Considering recent legistlative initiatives [14], we may be soon storing many verifiable credentials about our sensitive attributes in our smartphones, supported by electronic wallet applications, streamlining the procurement and presentation of these credentials. By themselves, smartphones cannot provide sufficient confidentiality for these credentials. Rather, we expect to store them in some encrypted form, decrypted only while they are in use. The decryption keys are stored inside a Secure Element [22], a tamper-resistant piece of hardware contained in the phone. Hence we have to trust the producers of Secure Elements, and abstain from the use of e-Wallets (with credentials containing sensitive information) on phones without Secure Elements. This is not a desirable situation. We would like to replace trusted hardware with something that has weaker trust requirements, e.g. threshold cryptography.

Common constructions of primitives of threshold cryptography and their security definitions are difficult to map to a setting where some keyshares are stored and operations performed on platforms with weak security protections. We have proposed a server-supported signature scheme [7], where the signing key was shared between a phone and a server, and the keyshare in the phone was protected only by symmetrically encrypting it with a key with very low entropy (derived from a PIN that the user can remember). The security of our scheme

G. Lenzini and W. Meng (Eds.): STM 2022, LNCS 13867, pp. 71–81, 2023.
https://doi.org/10.1007/978-3-031-29504-1_4

was based on the infeasibility of offline guessing attacks by someone who has obtained the encrypted keyshare of the phone, and by the ability of the server to recognize online guessing attacks. The latter property allows the server to count wrong guesses, and a clone detection mechanism [19] allows to reset the counter.

In this paper, we propose an encryption scheme with similar properties, i.e. it has distributed decryption, where offline guesses by someone masquerading as the phone are impossible, and wrong guesses made online are detected by the server. We want the phone to initiate the decryption, and the server to learn nothing about the decrypted plaintext. We give a formalization of these properties. Combined with the clone detection mechanism, our scheme could be used as an alternative to Secure Elements, *at least* when the requirement for online connectivity during decryption is acceptable.

Related Work. Our encryption scheme is motivated by a set of requirements that have previously not been tried to address together. They have been considered in the context of server-supported signature schemes [7], where we attempt to avoid *offline guessing attacks* [2] and detect *online guessing attacks* [12]. This is also the case for scheme [8], where the server supports the functionality of a secure construction. It is in contrast to the schemes where a server is employed to reduce the client's workload in performing computationally expensive operations [1,4]. It is also in contrast to *key-insulated encryption* [13], where a mostly offline server is used to reduce the impact of repeatedly breaking a weakly secure device.

Our scheme builds upon threshold encryption schemes. Threshold cryptography has a long history, starting from [11], where a method for threshold creation of RSA signatures was proposed. IND-CCA secure encryption schemes with threshold decryption [21] were proposed shortly after IND-CCA secure asymmetric encryption schemes [10]. At present, threshold cryptography is a mature field, discussed in textbooks [5] and subject to standardization activities [6].

2 Desired Properties of Distributed Decryption

In this paper, we consider asymmetric *key encapsulation* [18] schemes, where the decapsulation functionality is distributed between two parties—the *client*, and the *server*. The roles of these parties are not identical, and the desired security properties for each of them are different.

An encapsulation scheme with client-server decryption consists of the following sets, algorithms, and protocols, parameterized with the security parameter λ and other public parameters (e.g. the definition of the used cyclic groups):

- Sets of shared secrets SS, ciphertexts CT, public keys PK, client's private keys SK_C, and server's private keys SK_S.
- Key-generation protocol $\langle \mathcal{KG}_\mathrm{C} | \mathcal{KG}_\mathrm{S} \rangle$, run by both parties. It returns $(\mathsf{sk}_1, \mathsf{pk}) \in \mathsf{SK}_\mathrm{C} \times \mathsf{PK}$ to the client, and $(\mathsf{sk}_2, \mathsf{pk}) \in \mathsf{SK}_\mathrm{S} \times \mathsf{PK}$ to the server.
- Encapsulation algorithm $\mathcal{E}nc$. It takes as input a public key $\mathsf{pk} \in \mathsf{PK}$, and returns a shared secret $k \in \mathsf{SS}$ and a ciphertext $c \in \mathsf{CT}$.

Experiment IND-CCA-S$^{\mathcal{A}}$	Experiment IND-CCA-C$^{\mathcal{A}}$
$\langle(\mathsf{sk}_1, \mathsf{pk}), (\mathsf{sk}_2, \mathsf{state})\rangle \leftarrow\!\!\$\ \langle \mathcal{KG}_{\mathrm{C}} \vert \mathcal{A}()\rangle$	$\langle(\mathsf{sk}_1, \mathsf{state}), (\mathsf{sk}_2, \mathsf{pk})\rangle \leftarrow\!\!\$\ \langle \mathcal{A}() \vert \mathcal{KG}_{\mathrm{S}}\rangle$
$(k_0, c) \leftarrow\!\!\$\ \mathcal{E}nc(\mathsf{pk})$	$(k_0, c) \leftarrow\!\!\$\ \mathcal{E}nc(\mathsf{pk})$
$k_1 \leftarrow\!\!\$\ \mathsf{SS},\ b \leftarrow\!\!\$\ \{0,1\}$	$k_1 \leftarrow\!\!\$\ \mathsf{SS},\ b \leftarrow\!\!\$\ \{0,1\}$
$\mathcal{O}_1(\cdot) \leftarrow \mathcal{D}ec(\cdot, \mathsf{sk}_1, \mathsf{sk}_2, \mathsf{pk})$	$\mathcal{O}_1(\cdot) \leftarrow \mathcal{D}ec(\cdot, \mathsf{sk}_1, \mathsf{sk}_2, \mathsf{pk})$
$\mathcal{O}_2(\cdot) \leftarrow \mathcal{DC}_{\mathrm{C}}(\cdot, \mathsf{sk}_1, \mathsf{pk})$	$\mathcal{O}_2(\cdot) \leftarrow \mathcal{DC}_{\mathrm{S}}(\cdot, \mathsf{sk}_2, \mathsf{pk})$
$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{EXCL}_c[\mathcal{O}_1], \mathsf{EXCL}_c[\mathcal{O}_2]}(\mathsf{state}, c, k_b)$	$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{EXCL}_c[\mathcal{O}_1], \mathsf{EXCL}_c[\mathcal{O}_2]}(\mathsf{state}, c, k_b)$
return $b = b^*$	**return** $b = b^*$

Fig. 1. Security against chosen-ciphertext attacks

– Decapsulation protocol $\langle \mathcal{DC}_{\mathrm{C}} | \mathcal{DC}_{\mathrm{S}} \rangle$, run by the client and the server. Client's inputs are $c \in \mathsf{CT}$, $\mathsf{sk}_1 \in \mathsf{SK}_{\mathrm{C}}$, and $\mathsf{pk} \in \mathsf{PK}$. Server's inputs are $c \in \mathsf{CT}$, $\mathsf{sk}_2 \in \mathsf{SK}_{\mathrm{S}}$ and $\mathsf{pk} \in \mathsf{PK}$. The protocol returns either $k \in \mathsf{SS}$ or the failure notice $\perp$ to the client. It returns success/failure notice $\top/\perp$ to the server.

We also define the *decapsulation algorithm* $\mathcal{D}ec$, that on inputs $c, \mathsf{sk}_1, \mathsf{sk}_2, \mathsf{pk}$ invokes $\langle \mathcal{DC}_{\mathrm{C}}(c, \mathsf{sk}_1, \mathsf{pk}) | \mathcal{DC}_{\mathrm{S}}(\mathsf{sk}_2, \mathsf{pk}) \rangle$ and returns client's output.

In the following, we write $x_1, \ldots, x_n \leftarrow\!\!\$\ X$ to denote that values $x_1, \ldots, x_n$ are uniformly, independently sampled from a set X. We also write $x \leftarrow\!\!\$\ \mathsf{X}(\ldots)$ to denote that x is returned by a stochastic computation X. Given an oracle $\mathcal{O}(\cdot)$ and a value c, we let $\mathsf{EXCL}_c[\mathcal{O}]$ denote an oracle that on input c^* returns $\perp$ if $c^* = c$, and $\mathcal{O}(c^*)$ otherwise. A protocol party executed as an oracle gives the adversary the messages this party produces.

Definition 1 (Correctness). *A encapsulation scheme is* correct*, if*

$$\Pr\left[k' = k \wedge r = \top \;\middle|\; \begin{array}{l} \langle(\mathsf{sk}_1, \mathsf{pk}), (\mathsf{sk}_2, \mathsf{pk})\rangle \leftarrow\!\!\$\ \langle \mathcal{KG}_{\mathrm{C}} | \mathcal{KG}_{\mathrm{S}}\rangle, (k, c) \leftarrow \mathcal{E}nc(\mathsf{pk}), \\ \langle k', r\rangle \leftarrow \langle \mathcal{DC}_{\mathrm{C}}(c, \mathsf{sk}_1, \mathsf{pk}) | \mathcal{DC}_{\mathrm{S}}(c, \mathsf{sk}_2, \mathsf{pk})\rangle \end{array}\right] \approx 1.$$

The confidentiality properties of the encapsulation scheme are defined in the usual manner. The definitions refer to the experiments in Fig. 1 that follow general definitions of IND-CCA for threshold encryption schemes.

Definition 2 (IND-CCA against server/client). *The encapsulation scheme provides indistinguishability against the chosen-ciphertext attacks by the server [resp. client], if the experiment IND-CCA-S$^{\mathcal{A}}$ [resp. IND-CCA-C$^{\mathcal{A}}$] is successful with probability at most negligibly larger than* $1/2$ *for all efficient adversaries* $\mathcal{A}$*.*

The impossibility of offline guessing and detectability of online guessing is defined below, using the experiment defined in Fig. 2. Here $\mathsf{shuffle}$ returns a list that is a random permutation of its arguments. The list SK_1 corresponds to the list of candidate private keys that an intruder may obtain after extracting the weakly encrypted (e.g. the encryption key has been derived from a PIN) private

Experiment OG-CCA-C$^{\mathcal{A}}_{T,L}$

$\langle(\mathsf{sk}_1^{(1)}, \mathsf{pk}), (\mathsf{sk}_2, \mathsf{pk})\rangle \leftarrow\!\!\$\ \langle \mathcal{KG}_{\mathrm{C}} | \mathcal{KG}_{\mathrm{S}} \rangle$
$\mathsf{sk}_1^{(2)}, \ldots, \mathsf{sk}_1^{(L)} \leftarrow\!\!\$\ \mathsf{SK}_{\mathrm{C}},\ SK_1 \leftarrow\!\!\$\ \mathsf{shuffle}(\mathsf{sk}_1^{(1)}, \ldots, \mathsf{sk}_1^{(L)})$
$(k_0, c) \leftarrow\!\!\$\ \mathcal{E}nc(\mathsf{pk}),\ k_1 \leftarrow\!\!\$\ \mathsf{SS},\ b \leftarrow\!\!\$\ \{0,1\},\ t \leftarrow 0$
$\mathcal{O}_2(\cdot) \leftarrow \{\ t \leftarrow t+1;\ r \leftarrow \mathcal{DC}_{\mathrm{S}}(\cdot, \mathsf{sk}_2, \mathsf{pk});\ \mathbf{if}(r = \top)\ \mathbf{then}\ t \leftarrow t-1\ \}$
$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{EXCL}_c[\mathcal{D}ec(\cdot, \mathsf{sk}_1^{(1)}, \mathsf{sk}_2, \mathsf{pk})], \mathcal{O}_2(\cdot)}(\mathsf{pk}, c, k_b, SK_1)$
return $(b = b^*)$ and $t \leq T$

Fig. 2. Security against offline and online guessing

key from the smartphone, and trying to decrypt it with all possible values of the key. We see that the adversary may start sessions of the server, and may even submit it the challenge ciphertext, but no more than T sessions may finish with $\perp$ (or not finish at all).

Definition 3 (No guessing by client). *The encapsulation scheme provides offline guessing security against chosen-ciphertext attacks by the client, if the experiment OG-CCA-C$^{\mathcal{A}}_{T,L}$ is successful with probability at most negligibly larger than $1/2 + T/L$ for all efficient adversaries $\mathcal{A}$, and numbers T, L.*

Finally, we ask for the integrity of shared secrets, i.e. the client would not accept a secret k' different from the one output by the encapsulation algorithm.

Definition 4 (Integrity for client). *The encapsulation scheme provides key integrity for the client, if for all efficient adversaries $\mathcal{A}$,*

$$\Pr\left[k' \in \{k, \perp\} \;\middle|\; \begin{array}{l} \langle(\mathsf{sk}_1, \mathsf{pk}), \mathsf{state}\rangle \leftarrow \langle \mathcal{KG}_{\mathrm{C}} | \mathcal{A}() \rangle, (k, c) \leftarrow \mathcal{E}nc(\mathsf{pk}), \\ \langle k', \_ \rangle \leftarrow \langle \mathcal{DC}_{\mathrm{C}}(c, \mathsf{sk}_1, \mathsf{pk}) | \mathcal{A}(\mathsf{state}) \rangle \end{array}\right] \approx 1.$$

3 Building Blocks

Let $\mathbb{G}$ be a cyclic group of size p, with generator g. The *discrete logarithm problem* is to find $n \in \mathbb{Z}_p$, such that $g^n = h$, for a value $h \leftarrow\!\!\$\ \mathbb{G}$. The *decisional Diffie-Hellman (DDH) problem* is to distinguish tuples of the form (g, g^x, g^y, g^{xy}) (called *Diffie-Hellman tuples*) from tuples of the form (g, g^x, g^y, g^z) for $x, y, z \leftarrow\!\!\$\ \mathbb{Z}_p$. A problem is *hard* if all efficient algorithms have at most negligible advantage (over a trivial algorithm) of solving it.

Our schemes build on top of the ElGamal KEM, the IND-CPA security of which is equivalent to the hardness of DDH in the used group $\mathbb{G}$. In this KEM, private key is a random $\mathsf{sk} \in \mathbb{Z}_p$, while public key is $\mathsf{pk} = g^{\mathsf{sk}}$. The encapsulation algorithm generates $r \leftarrow\!\!\$\ \mathbb{Z}_p$, and outputs the shared secret $ss \leftarrow \mathsf{pk}^r$ and ciphertext $c = g^r$. The decapsulation algorithm computes $ss = c^{\mathsf{sk}}$. In *hashed ElGamal*, the shared secret is $H'(\mathsf{pk}^r)$ for some hash function H' that we model as a random oracle. Note that an input to a random oracle can be anything encodable as a bitstring.

A *DDH proof* $\pi = (\alpha, \alpha', \gamma) \leftarrow_\$ \mathsf{DHP}^H[r \,|\, {}^{u,v}_{g,h} | ctx]$ [9] is a non-interactive [16] zero-knowledge (NIZK) proof that $\log_g u = \log_h v$, given in *context* ctx, where $r \in \mathbb{Z}_p$ is the discrete logarithm and H is a hash function, modeled as a random oracle. It is given by $s \leftarrow_\$ \mathbb{Z}_p$, $\alpha \leftarrow g^s$, $\alpha' \leftarrow h^s$, $\beta \leftarrow H(g, h, u, v, \alpha, \alpha', ctx) \in \mathbb{Z}_p$, and $\gamma \leftarrow s + r \cdot \beta$. The checking procedure $\mathsf{ChP}^H[\pi \,|\, {}^{u,v}_{g,h} | ctx]$ recomputes β, and checks that $g^\gamma = \alpha \cdot u^\beta$ and $h^\gamma = \alpha' \cdot v^\beta$.

In our schemes, similarly to [21], DDH proofs are often used to give *simulatable proofs of knowledge of exponent* $\pi' \leftarrow_\$ \mathsf{KnE}^{H,\tilde{H}}[r \,|\, {}^{u}_{g} | ctx]$. These prove that someone knows the value $r = \log_g u$. Additionally, they allow the simulator to raise a value (an element of $\mathbb{G}$) of its choice to the power of r; the simulator has to choose that value at the time the adversary computes the proof. The construction makes use of two hash functions, both modeled as random oracles, where H returns elements of $\mathbb{Z}_p$ and $\tilde{H}$ returns elements of $\mathbb{G}$. It is given by first computing $h \leftarrow \tilde{H}(g, u, ctx)$ and $v \leftarrow h^r$. The proof is $\pi' = (\pi, v)$, where $\pi \leftarrow \mathsf{DHP}^H[r \,|\, {}^{u,v}_{g,h} | ctx]$. The checking procedure $\mathsf{ChE}^{H,\tilde{H}}[\pi' \,|\, {}^{u}_{g} | ctx]$ recomputes v and checks the DDH proof. During simulation, if the simulator wants to obtain z^r, it will generate $t \leftarrow_\$ \mathbb{Z}_p$, and program $\tilde{H}$ to return $z^{1/t}$ when the adversary queries it with g, u, ctx. Then $z^r = v^t$.

4 The Encryption Scheme

Secret-sharing the private key will straightforwardly thresholdize the ElGamal KEM [15]. IND-CCA may be achieved by adding the *non-interactive zero-knowledge proof of knowledge* (NIZKPoK) of r to the ciphertext. For non-threshold systems, this may be a designated-verifier (DV) NIZKPoK [10], aimed towards the receiver. For general case, Schnorr's proofs for discrete logarithm [20], made non-interactive through the Fiat-Shamir transform [16] using a random oracle (i.e. Schnorr signatures, using r as the signing key), are typically used to show the knowledge of r, but it is unknown how to combine them with ElGamal KEM in a way that allows IND-CCA to be derived only from the hardness of the DDH problem [3]. The TDH2 (threshold) cryptosystem [21] overcomes this by changing how the random oracle is used by the Fiat-Shamir transform, making certain additional computations possible in the simulation. The scheme NPS that we present here is quite similar to TDH2. Interestingly, only small changes are needed to make it secure against guessing attacks (Definition 3).

Let $\mathbb{G}$ be a cyclic group of size p, with generator g, with hard DDH problem. Let H_1, H_2, H_3 be hash functions outputting elements of $\mathbb{Z}_p$, and $\tilde{H}_1, \tilde{H}_2$ be hash functions outputting elements of $\mathbb{G}$, all modeled as random oracles. We put $\mathsf{NPS.SS} = \mathsf{NPS.PK} = \mathbb{G}$, $\mathsf{NPS.SK_C} = \mathbb{Z}_p$, and $\mathsf{NPS.SK_S} = \mathbb{Z}_p \times \mathbb{G}^2$. The set $\mathsf{NPS.CT}$ is given together with the algorithm $\mathsf{NPS}.\mathcal{E}nc$ and protocol $\mathsf{NPS}.\mathcal{DC}$ in Fig. 3. In the key-generation protocol, the client generates $\mathsf{sk}_1 \leftarrow_\$ \mathbb{Z}_p$, and the server generates $\mathsf{sk}_2 \leftarrow_\$ \mathbb{Z}_p$. They compute $\mathsf{pk}_i \leftarrow g^{\mathsf{sk}_i}$, fairly exchange the values pk_i with each other (using some *trapdoor commitment* scheme [17]), and define $\mathsf{pk} \leftarrow \mathsf{pk}_1 \cdot \mathsf{pk}_2$. The server stores $\mathsf{pk}_1, \mathsf{pk}_2$ together with the private exponent sk_2.

$\mathsf{NPS}.\mathcal{E}nc(\mathsf{pk})$

$r \leftarrow\!\!\$\ \mathbb{Z}_p$
$u \leftarrow g^r$
$\pi \leftarrow\!\!\$\ \mathsf{KnE}^{H_1,\tilde{H}_1}[r \mid {}^{u}_{g}]$
return $\mathsf{pk}^r, (u, \pi)$

$\mathsf{NPS}.\mathcal{DC}_{\mathrm{C}}((u,\pi), \mathsf{sk}_1, \mathsf{pk})$

$\mathsf{ChE}^{H_1,\tilde{H}_1}[\pi \mid {}^{u}_{g}]$
$\mathsf{pk}_1 \leftarrow g^{\mathsf{sk}_1}, \mathsf{pk}_2 \leftarrow \mathsf{pk}/\mathsf{pk}_1$
$\pi' \leftarrow\!\!\$\ \mathsf{KnE}^{H_2,\tilde{H}_2}[\mathsf{sk}_1 \mid {}^{\mathsf{pk}_1}_{g} \mid u]$
$\longrightarrow P_2 : u, \pi, \pi'$
$P_2 \longrightarrow : w, \pi''$
$\mathsf{ChP}^{H_3}[\pi'' \mid {}^{u,w}_{g,\mathsf{pk}_2} \mid \pi']$
return $u^{\mathsf{sk}_1} \cdot w$

$\mathsf{NPS}.\mathcal{DC}_{\mathrm{S}}((u,\pi), \mathsf{sk}_2, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk})$

$P_1 \longrightarrow : u, \pi, \pi'$
$\mathsf{ChE}^{H_1,\tilde{H}_1}[\pi \mid {}^{u}_{g}]$
$\mathsf{ChE}^{H_2,\tilde{H}_2}[\pi' \mid {}^{\mathsf{pk}_1}_{g} \mid u]$
$w \leftarrow u^{\mathsf{sk}_2}$
$\pi'' \leftarrow\!\!\$\ \mathsf{DHP}^{H_3}[\mathsf{sk}_2 \mid {}^{u,w}_{g,\mathsf{pk}_2} \mid \pi']$
$\longrightarrow P_1 : w, \pi''$
return $\top$

Fig. 3. Encryption and decryption for the scheme NPS

We see that the ciphertext is simply an ElGamal ciphertext, together with a simulatable proof of knowledge of the exponent r. When decrypting, the client first verifies this proof. If an assertion fails, then $\perp$ is immediately returned. As next, the client asks the server to apply its private key share sk_2 to the ciphertext u. This request includes a Schnorr proof of knowing the private key sk_1, where the challenge depends on the ciphertext; hence this proof cannot be reused. The request also contains the value h'' that allows the simulator to perform an exponentiation with sk_1. The server verifies the Schnorr proofs of knowing both r and sk_1, and then computes $w \leftarrow u^{\mathsf{sk}_2}$. The value w is returned together with a Schnorr proof that it has been correctly computed—that $\log_u w = \log_g \mathsf{pk}_2$. The client verifies this proof, applies its private key share sk_1 to u, and combines the result with the plaintext share w obtained from the server. It is clear that the scheme satisfies Definition 1 and Definition 4 due to the NIZK proofs.

Theorem 1. *In ROM, if the DDH problem is hard in group $\mathbb{G}$, then* NPS *provides IND-CCA against the server and the client.*

Proof. To show IND-CCA against the server, let $\mathcal{A}$ be an adversary that has non-negligible advantage in experiment IND-CCA-S$^{\mathcal{A}}$ with the scheme NPS. We construct an algorithm $\mathcal{S}$ that solves the DDH problem in $\mathbb{G}$. The algorithm $\mathcal{S}$ (called "simulator") internally calls $\mathcal{A}$, realizing the oracles it accesses, including the random oracle. It receives $(h_1, h_2, h_3) \in \mathbb{G}^3$ as an input, and outputs whether they are a DH tuple. In the experiment, the values h_i play the following roles: $\mathsf{pk} = h_1$, $u = h_2$, $k_b = h_3$. We see that if (g, h_1, h_2, h_3) are [resp. are not] a DH tuple, then (pk, u, k_b) are distributed identically to the case $b = 0$ [resp. $b = 1$].

The internal state of $\mathcal{S}$ contains the tables $\mathcal{T}_i, \tilde{\mathcal{T}}_j$ storing the current states of random oracles H_i, $\tilde{H}_j$. For tables $\mathcal{T}_i$, a table row contains the argument made to the oracle, and the given response. For $\tilde{\mathcal{T}}_j$, a table row additionally contains the exponent generated while responding a $\tilde{H}_j$-query.

We now describe how $\mathcal{S}$ behaves in different interactions with $\mathcal{A}$. For key generation, receive pk_2 (committed and opened) from $\mathcal{A}$, while sending it a commitment later opened to $\mathsf{pk}_1 \leftarrow h_1/\mathsf{pk}_2$. For responding a random oracle query (either directly from $\mathcal{A}$, or from simulating the responses to other queries)

$H_i(x)$, look up the row (x, v) from $\mathcal{T}_i$, generating $v \leftarrow\!\!\$\, \mathbb{Z}_p$ and adding the row to the table, if it is not there. Then respond with v. Do the same for $\tilde{H}_2$-query (put $\perp$ as the exponent). Also do the same for query $\tilde{H}_j(x)$, but if the row (x, v, t) is not yet in $\tilde{\mathcal{T}}_j$ then generate $t \leftarrow\!\!\$\, \mathbb{Z}_p$, define $v = \mathsf{pk}^{1/t}$, add (x, v, t) to $\tilde{\mathcal{T}}_1$, and return v.

Simulator $\mathcal{S}$ has to prepare a challenge ciphertext $c = (u, h', \alpha, \alpha', \gamma)$ for $\mathcal{A}$. We have defined u; the rest is constructed by faking the NIZK proofs: $\mathcal{S}$ generates $s \leftarrow\!\!\$\, \mathbb{Z}_p$, defines $g' \leftarrow g^s$, $h' \leftarrow h_2^s$, generates $\beta, \gamma \leftarrow\!\!\$\, \mathbb{Z}_p$, computes $\alpha \leftarrow u^\beta / g^\gamma$ and $\alpha' \leftarrow (h')^\beta / (g')^\gamma$, and adds $((g', u, h', \alpha, \alpha'), \beta)$ to $\mathcal{T}_1$ and $((u, \alpha), g', \perp)$ to $\tilde{\mathcal{T}}_1$. This computation may fail if the added rows are already present in the tables, but this happens with only negligible probability because all arguments contain fresh randomness.

A query from $\mathcal{A}$ to the $\mathcal{D}ec$-oracle with argument $c^* = (u^*, h'^*, \alpha^*, \alpha'^*, \gamma^*)$ is handled by $\mathcal{S}$ as follows. First check the proofs, similarly to $\mathsf{NPS}.\mathcal{DC}_\mathrm{C}$. Return $\perp$, if they fail. Otherwise look up the row $((u^*, \alpha^*), g', t)$ in $\tilde{\mathcal{T}}_1$ and return $(h'^*)^t$. This row has to exist, because the proof-checks look it up. The value t may be missing, but in this case c^* had to be the challenge ciphertext.

A query from $\mathcal{A}$ to the $\mathcal{DC}_\mathrm{C}$-oracle with the same argument c^* is handled by $\mathcal{S}$ as follows. Check the proofs; return $\perp$ if they fail. Prepare the query to the server, faking the proof of knowledge π' of sk_1. Whenever $\mathcal{A}$ invokes $\mathcal{DC}_\mathrm{C}$ for the second round, $\mathcal{S}$ ignores this query: the server is not expected to get any answer from the client's second round.

Throughout this construction, all values that $\mathcal{A}$ sees are distributed identically to the experiment IND-CCA-S$^{\mathcal{A}}$ for NPS. In particular, the responses from the random oracles are uniform and mutually independent. Finally, $\mathcal{A}$ gives its guess b^*, which $\mathcal{S}$ outputs. The advantage of $\mathcal{S}$ is equal to the advantage of $\mathcal{A}$.

The proof of IND-CCA against client is similar. When the $\mathcal{A}$ generates the simulatable proof of knowledge of sk_1 (as in $\mathcal{DC}_\mathrm{C}$) and invokes $\tilde{H}_2(g, \mathsf{pk}_1, u)$ for that purpose, $\mathcal{S}$ chooses u as the value it wants to raise to power sk_1. If $\mathcal{A}$ then invokes the oracle $\mathcal{DC}_\mathrm{S}$, the simulator $\mathcal{S}$ can reply with $w \leftarrow \mathsf{pk}^r / u^{\mathsf{sk}_1}$. □

Theorem 2. *In ROM, if the DDH problem is hard in group $\mathbb{G}$, then* NPS *provides CCA against offline guessing attacks by the client.*

Proof. Similarly to the proof of Theorem 1, we assume the existence of an adversary $\mathcal{A}$ that has advantage at least $1/2 + T/L + \nu$ for a non-negligible ν in the experiment OG-CCA-C$^{\mathcal{A}}_{T,L}$ with the scheme NPS, and construct the algorithm $\mathcal{S}$ that solves the DDH problem in $\mathbb{G}$. It again gets (h_1, h_2, h_3) as the input, and again uses them as $\mathsf{pk} = h_1$, $u = h_2$, $k_b = h_3$.

The internal state of $\mathcal{S}$ consists of the same tables as in the proof of Theorem 1. For some $\mathsf{h} \in \mathbb{G}$, we additionally define $\mathcal{T}_2|_\mathsf{h}$ as the subset of rows of $\mathcal{T}_2$ of the form $((\mathsf{h}, \alpha_1, c), v)$ (i.e. the first component to the argument of H_2 was h). The simulator $\mathcal{S}$ also maintains a set of integers $\mathbf{K}$, initialized to $\{1, \ldots, L\}$.

Simulator $\mathcal{S}$ has to prepare the arguments to $\mathcal{A}$. The challenge ciphertext $c = (u, h', \alpha, \alpha', \gamma)$ is prepared identically to the proof of Theorem 1. The list of potential private key shares of the client is defined by generating

$\mathsf{sk}_1^{(1)}, \ldots, \mathsf{sk}_1^{(L)} \leftarrow_\$ \mathbb{Z}_p$ and putting $SK_1 = [\mathsf{sk}_1^{(1)}, \ldots, \mathsf{sk}_1^{(L)}]$. Note that the inputs to $\mathcal{A}$ are distributed identically to the experiment OG-CCA-C$^{\mathcal{A}}_{T,L}$ for NPS. Also note that at this point, $\mathcal{S}$ has not selected the "right" private key share. Define also $\mathsf{pk}_1^{(i)} \leftarrow g^{\mathsf{sk}_1^{(i)}}$ for $i \in \{1, \ldots, L\}$.

A query from $\mathcal{A}$ to either one of the hash functions or to the $\mathcal{D}ec$-oracle is handled identically to the proof of Theorem 1. Again, all responses to these queries are distributed identically to the actual experiment OG-CCA-C$^{\mathcal{A}}_{T,L}$.

The $\mathbf{t}$-th query $(c_\mathbf{t}, \alpha_{1,\mathbf{t}}, \gamma_{1,\mathbf{t}})$ to the $\mathcal{DC}_\mathrm{S}$-oracle is handled by $\mathcal{S}$ as follows. First, verify the proofs in $c_\mathbf{t} = (u_\mathbf{t}, h'_\mathbf{t}, \alpha_\mathbf{t}, \alpha'_\mathbf{t}, \gamma_\mathbf{t})$, corresponding to the assertions in NPS.$\mathcal{DC}_\mathrm{S}$. Return $\perp$, if these verifications fail.

The simulator continues with the response for $\mathcal{DC}_\mathrm{S}$-oracle as follows. It finds the index $i \in \{1, \ldots, L\}$, such that the row $((\mathsf{pk}_1^{(i)}, \alpha_{1,\mathbf{t}}, \gamma_{1,\mathbf{t}}), \beta_{1,\mathbf{t}})$ is in the table $\mathcal{T}_2$ for some $\beta_{1,\mathbf{t}}$, and $g^{\gamma_{1,\mathbf{t}}} = \alpha_{1,\mathbf{t}} \cdot (\mathsf{pk}_1^{(i)})^{\beta_{1,\mathbf{t}}}$. If there is no such row, then let $i = \perp$. The probability of having more than one such row in $\mathcal{T}_2$ is negligible. Indeed, if i and i' would both be such indices, then $(\mathsf{pk}_1^{(i)})^{\beta_{1,\mathbf{t}}} = (\mathsf{pk}_1^{(i')})^{\beta'_{1,\mathbf{t}}}$. The values $\beta_{1,\mathbf{t}}$ and $\beta'_{1,\mathbf{t}}$ are random, and generated independently from $\mathsf{pk}_1^{(i)}$ and $\mathsf{pk}_1^{(i')}$, hence this equality can hold only with negligible probability.

If $i \notin \mathbf{K}$, then $\mathcal{S}$ returns $\perp$ to $\mathcal{A}$. If $i \in \mathbf{K}$, then $\mathcal{S}$ has to decide whether the "right" private key is $\mathsf{sk}_1^{(i)}$. For this purpose, $\mathcal{S}$ tosses a biased coin, with the result "heads" having the probability $1/|\mathbf{K}|$. If the result is "heads", then this means that the "right" private key was indeed $\mathsf{sk}_1^{(i)}$. In this case, $\mathcal{S}$ gives up the simulation, outputting $\perp$. Otherwise, $\mathcal{S}$ sets $\mathbf{K} \leftarrow \mathbf{K} \backslash \{i\}$ and returns $\perp$ to $\mathcal{A}$.

Again, we have that as long as $\mathcal{A}$ has not managed to find the "right" private key, all values in the simulation are distributed identically to the experiment OG-CCA-C$^{\mathcal{A}}_{T,L}$ for NPS. The probability of finding the "right" private key is upper-bounded by T/L, hence $\mathcal{S}$ still has at least the non-negligible advantage ν in solving the DDH problem in $\mathbb{G}$. □

5 Fit for Our Main Use-Case

In our main use-case, the client is a smartphone, receiving and storing encrypted messages (e.g. credentials), and decrypting them for short uses. The smartphone communicates with the helper server over mobile internet, through a secure channel. An attacker in this system may have the following goals: (A) learn the plaintext corresponding to a ciphertext, or (B) make the phone accept wrong plaintext. Against this attacker we deploy NPS, including the clone detection. The latter may be continuous.

We consider the following attacks that an attacker may perform: (1) convince the phone to start decryption protocol with a particular ciphertext; (2) learn phone's encrypted memory; (3) learn phone's unencrypted memory; (4) learn server's keyshare; (5) take passive control over server; (6) take active control over server; (7) masquerade as phone to the server; (8) masquerade as server to the phone.

Let the boolean variable x indicate whether the clone detection is done continuously. Let y_i ($i \in \{1, \ldots, 8\}$) be a boolean variable indicating whether adversary has successfully performed the attack (i); note that $y_3 \Rightarrow y_2$, $y_6 \Rightarrow y_5$, and $y_5 \Rightarrow y_4$. Let the boolean variables z_{X} for $\mathrm{X} \in \{\mathrm{A}, \mathrm{B}\}$ indicate that the adversary achieves his goal X. Let also $\tilde{z}_{\mathrm{X}}$ denote that the goal is achieved only for a short time (until the continuously running clone detection mechanism discovers something); obviously $z_{\mathrm{X}} \Rightarrow \tilde{z}_{\mathrm{X}}$. The following implications hold:

$$[y_2 \wedge y_4] \vee [((y_2 \wedge y_8) \vee y_3) \wedge y_7 \wedge \neg x] \Rightarrow z_{\mathrm{A}} \qquad ((y_2 \wedge y_8) \vee y_3) \wedge y_7 \Rightarrow \tilde{z}_{\mathrm{A}}$$

Also, if some z_{X} or $\tilde{z}_{\mathrm{X}}$ is true, then this must follow from some of the implications. We see that z_{B} never holds, because Definition 4 forbids it. Server's keyshare and phone's encrypted keyshare are sufficient for decryption. So is the knowledge of phone's unencrypted keyshare, if the attacker can masquerade the phone and the clone detection does not stop the attack. Interestingly, $y_2 \wedge y_8 \Rightarrow y_3$, because the information sent in phone's first message may enable the offline guessing attack against the PIN. We consider all these vulnerabilities acceptable.

We see that our scheme adds significant overhead to "plain" IND-CPA secure ElGamal. We also see that the overheads are wholly acceptable for our main intended use-case. We have implemented NPS encryption and decryption in Python on top of the PyCryptodome cryptographic library, using the elliptic curve group P-256 as $\mathbb{G}$; the running times are 6 ms for $\mathcal{E}nc$, 135 ms for $\mathcal{DC}_{\mathrm{C}}$, and 107 ms for $\mathcal{DC}_{\mathrm{S}}$ running on a laptop with an Intel® Core™ i5-10210U CPU and 16GB RAM. The size of a key encapsulation is 6 kilobytes, the messages sent from the client to the server and back: 6.7KB and 5.5KB, respectively.

Acknowledgement. This research has been funded by the European Regional Development Fund through EXCITE, the Estonian Centre of Excellence in ICT.

References

1. Asokan, N., Tsudik, G., Waidner, M.: Server-supported signatures. J. Comput. Secur. **5**(1), 91–108 (1997). https://doi.org/10.3233/JCS-1997-5105
2. Baudet, M.: Deciding security of protocols against off-line guessing attacks. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, pp. 16–25. Association for Computing Machinery, New York (2005). https://doi.org/10.1145/1102120.1102125
3. Bernhard, D., Fischlin, M., Warinschi, B.: On the hardness of proving CCA-security of signed ElGamal. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 47–69. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_3
4. Bicakci, K., Baykal, N.: Server assisted signatures revisited. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 143–156. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24660-2_12
5. Boneh, D., Shoup, V.: A Graduate Course in Applied Cryptography (2020). A book in preparation, v0.5

6. Brañdao, L.T.A.N., Mouha, N., Vassilev, A.: Threshold schemes for cryptographic primitives. Technical report. NISTIR 8214, National Institute of Standards and Technology (NIST) (2019)
7. Buldas, A., Kalu, A., Laud, P., Oruaas, M.: Server-supported RSA signatures for mobile devices. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10492, pp. 315–333. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66402-6_19
8. Buldas, A., Laanoja, R., Truu, A.: A server-assisted hash-based signature scheme. In: Lipmaa, H., Mitrokotsa, A., Matulevičius, R. (eds.) NordSec 2017. LNCS, vol. 10674, pp. 3–17. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70290-2_1
9. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_7
10. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055717
11. De Santis, A., Desmedt, Y., Frankel, Y., Yung, M.: How to share a function securely. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, STOC 1994, pp. 522–533. Association for Computing Machinery, New York (1994). https://doi.org/10.1145/195058.195405
12. Ding, Y., Horster, P.: Undetectable on-line password guessing attacks. ACM SIGOPS Oper. Syst. Rev. **29**(4), 77–86 (1995). https://doi.org/10.1145/219282.219298
13. Dodis, Y., Katz, J., Xu, S., Yung, M.: Key-insulated public key cryptosystems. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 65–82. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_5
14. Proposal for a Regulation of the European Parliament and of the Council amending Regulation (EU) No 910/2014 as regards establishing a framework for a European Digital Identity (SEC(2021) 228 final) - (SWD(2021) 124 final) - (SWD(2021) 125 final) (2021). https://digital-strategy.ec.europa.eu/en/library/trusted-and-secure-european-e-id-regulation
15. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_2
16. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
17. Fischlin, M.: Trapdoor commitment schemes and their applications. Ph.D. thesis, Goethe University Frankfurt, Frankfurt am Main, Germany (2001). https://zaurak.tm.informatik.uni-frankfurt.de/diss/data/src/00000229/00000229.pdf.gz
18. ISO 18033-2: Encryption algorithms—Part 2: Asymmetric ciphers. Standard, International Organization for Standardization (2006)
19. Sarr, A.P.: Cryptanalysis and improvement of smart-ID's clone detection mechanism. Cryptology ePrint Archive, Paper 2019/1412 (2019). https://eprint.iacr.org/2019/1412
20. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_22

21. Shoup, V., Gennaro, R.: Securing threshold cryptosystems against chosen ciphertext attack. J. Cryptol. **15**(2), 75–96 (2002). https://doi.org/10.1007/s00145-001-0020-9
22. Vauclair, M.: Secure element. In: van Tilborg, H.C.A., Jajodia, S. (eds.) Encyclopedia of Cryptography and Security, pp. 1115–1116. Springer, Boston (2011). https://doi.org/10.1007/978-1-4419-5906-5_303