Forward-Secure Distributed Encryption*

Wouter Lueks[1], Jaap-Henk Hoepman[1], and Klaus Kursawe[2]

[1] Radboud University Nijmegen, Nijmegen, The Netherlands
{lueks,jhh}@cs.ru.nl
[2] The European Network for Cyber Security, The Hague, The Netherlands
klaus.kursawe@ecns.eu

Abstract. Distributed encryption is a cryptographic primitive that implements revocable privacy. The primitive allows a recipient of a message to decrypt it only if enough senders encrypted that same message. We present a new distributed encryption scheme that is simpler than the previous solution by Hoepman and Galindo—in particular it does not rely on pairings—and that satisfies stronger security requirements. Moreover, we show how to achieve key evolution, which is necessary to ensure scalability in many practical applications, and prove that the resulting scheme is forward secure. Finally, we present a provably secure batched distributed encryption scheme that is much more efficient for small plaintext domains, but that requires more storage.

1 Introduction

Revocable privacy [6,14] has been proposed as a means for balancing security and privacy. A system implements revocable privacy if "the architecture of the system guarantees that personal data is revealed only if a predefined rule has been violated" [6]. For an in-depth discussion of the complex interactions between security and privacy, and the value of revocable privacy therein we refer to [6].

The distributed encryption scheme proposed by Hoepman and Galindo [7] is a primitive that can be used to implement revocable privacy. In particular it can be used to implement the rule "if a person or an object generates more than k events, its identity should be revealed." To do so, senders in the distributed encryption scheme encrypt the corresponding identity for every event that occurs. The scheme guarantees that the recipient of these ciphertexts can recover the identity only if it can combine k ciphertext shares, i.e., encryptions, of the same message created by different senders. We refer to [9] for other applications, but for the purpose of this paper we will examine the following two in more detail:

1. Consider the notarized sale of valuable objects like houses. Objects that change hands frequently may indicate fraud or money laundering and may therefore be

* This research is supported by the research program Sentinels as project 'Revocable Privacy' (10532). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs.
This research is conducted within the Privacy and Identity Lab (PI.lab) and funded by SIDN.nl (http://www.sidn.nl).

suspicious. A distributed encryption scheme can be used to identify the suspicious sales, while learning nothing about the others. To do so, every notary encrypts a record of every transaction under the distributed encryption scheme and submits the ciphertext share to a central authority. The authority can then recover only the details of the suspicious objects.

2. A distributed encryption scheme can enforce speed limits in a privacy-friendly manner as follows. Place automatic number plate recognition (ANPR) systems at the start and end of a stretch of highway, like in the SPECS system [13]. Suppose that a car is speeding if it takes at most t seconds to traverse this stretch. The ANPR systems generate ciphertext shares for every passing car. The system restarts after time t, so speeding cars generate two shares instead of one. They can be detected with a distributed encryption scheme with two senders and threshold one. To reliably detect speeding cars, the detection system needs to run multiple, staggered instances of the distributed encryption scheme. The higher the number of parallel instances, the better the accuracy. See Sect. 7 for the details.

The second application is especially challenging: the time frames are short and the number of observations is high. In this paper we propose two new schemes that can deal with these situations much more efficiently than the distributed encryption scheme by Hoepman and Galindo [7].

In the speed-limiting application, two-way interaction between the cars and ANPR systems is infeasible: adding communication facilities to cars would be costly. Therefore, distributed encryption schemes should be non-interactive to offer privacy in these situations. Techniques based on k-times anonymous credentials [2] and threshold encryption schemes—see [7] for a detailed discussion—are thus not appropriate. Also, when using a distributed encryption scheme the senders can immediately encrypt their observations. Storing a plaintext copy, as would be needed for a secure multi-party computation between the senders, is therefore not necessary. We see this as an advantage. The non-interactivity does imply, however, that the senders have to be trusted, and that framing is possible otherwise.

Our first contribution is a simpler distributed encryption scheme that does not use pairings, and satisfies stronger security requirements than the original scheme by Hoepman and Galindo [7]. We present this scheme in Sect. 4. We extend it with a non-trivial key-evolution method [5] to forward-securely [8] generate as many keys as necessary while keeping the key-size constant, see Sect. 5. The ability to restart the system with fresh keys is important in almost all applications, including the speed-limiting example, because it ensures that only shares generated within the same time frame can be combined. Hoepman and Galindo's original solution requires that the keys for every time frame are generated in advance, and therefore scales less well.

Our second contribution is a batched distributed encryption scheme. It addresses the issue of inefficiency in traditional distributed encryption schemes in practice. In the speed-limiting use case, for example, the cost to recover all encrypted plaintexts from a set of ciphertexts shares is exponential: the only option is to try all possible combinations of shares. Our batched solution, which we present in Sect. 6, is much more efficient, at the cost of increased storage requirements. The amount of storage is linear in the number of plaintexts. Hence this solution is feasible only if the domain is

small, as is the case for license plates. Nevertheless, we believe this to be a worthwhile trade-off.

Finally, in Sect. 7 we analyze the performance of our schemes, suggest additions to our scheme that may be useful in practice and present conclusions.

2 The idea

The idea of our new distributed encryption scheme is that a ciphertext share can be generated by first encoding the plaintext, and then transforming the ciphertext into a ciphertext share. If enough of these ciphertext shares are collected, the combiner can recover the plaintext.

Let $\mathbf{G}$ be a cyclic group of prime order q, such that DDH is hard in $\mathbf{G}$. The protocol uses an injection $\chi : \{0,1\}^{\ell_p} \to \mathbf{G}$ to encode plaintexts elements into group elements. The function χ^{-1} is the inverse of χ, i.e., $\chi^{-1}(\chi(p)) = p$. This mapping is redundant, i.e., with high probability a random group element $g \in \mathbf{G}$ has no inverse under χ^{-1}. We construct this map in the next section.

Every sender i is given a secret share $s_i \in \mathbf{Z}_q$, corresponding to a k-out-of-n Shamir secret sharing of a *publicly known* value, 1. Let f be the corresponding degree $k-1$ secret-sharing polynomial, i.e., $f(0) = 1$ and $s_i = f(i)$. Sender i produces a ciphertext share for plaintext p as follows. First, it encodes the plaintext into a generator $\chi(p) \in \mathbf{G}$. Then, it uses its secret share to produce the ciphertext share $\alpha_i = \chi(p)^{s_i}$. Given enough of these shares for the same plaintext, the exponents can be removed, and the original ciphertext can be recovered. More precisely, consider a set $\{\alpha_{i_1}, \ldots, \alpha_{i_k}\}$ of shares with $I = \{i_1, \ldots, i_k\}$ the set of indices, then there exist Lagrange coefficients $\lambda^I_{i_1}, \ldots, \lambda^I_{i_k}$ such that $\sum_{i \in I} \lambda^I_i s_i = f(0) = 1$. So, we can calculate

$$\alpha = \prod_{i \in I} \alpha_i^{\lambda^I_i} = \chi(p)^{\sum_{i \in I} s_i \lambda^I_i} = \chi(p)^{f(0)} = \chi(p).$$

Then, $p = \chi^{-1}(\alpha)$. If the shares do not belong to the same plaintext the resulting encoding will, with high probability not be an encoding of a plaintext element, and therefore fail to decode using χ^{-1}.

If χ is not redundant, the scheme is insecure. Let $p = \chi^{-1}(g)$ and $p' = \chi^{-1}(g^2)$. Then given a share $\alpha_i = \chi(p)^{s_i} = g^{s_i}$ it is trivial to make a share $\alpha'_i = \chi(p')^{s_i} = g^{2s_i} = (g^{s_i})^2$ for p' without help of the sender. As will become clear later, this breaks the scheme.

3 Preliminaries

We first recall some definitions. The security of our new distributed encryption schemes requires the following problem to be hard.

Definition 1 (Decisional Diffie-Hellman problem). *The Decisional Diffie-Hellman problem (DDH) in a group* $\mathbf{G}$ *of order q takes as input a tuple* $(g, A = g^a, B = g^b, C = g^c) \in \mathbf{G}^4$ *and outputs 'yes' if* $c = ab \pmod q$*, and else 'no.'*

We define Lagrange coefficients as used in Shamir's secret sharing scheme [12].

Definition 2 (Lagrange coefficients). *For a set $I \subseteq \{1,\ldots,n\}$ and field $\mathbf{Z}_q$ with $q > n$, we define the* Lagrange polynomials *$\lambda_i^I(x)$ as $\lambda_i^I(x) = \prod_{t\in I\setminus\{i\}} \frac{x-t}{i-t} \in \mathbf{Z}_q^*[x]$, and the* Lagrange coefficients *as $\lambda_i^I = \lambda_i^I(0)$. Then, for any polynomial $P \in \mathbf{Z}_q[x]$ of degree at most $|I|-1$, $P(x) = \sum_{i\in I} P(i)\lambda_i^I(x)$ and $P(0) = \sum_{i\in I} P(i)\lambda_i^I$.*

Notation. We write $|A|$ to denote the cardinality of the set A, $[n]$ to denote the set $\{1,\ldots,n\}$ and $x \parallel y$ to denote the concatenation of the strings x and y. Finally, $x \in_R A$ denotes that x is drawn uniformly at random from the set A.

Redundant Injective Map. We now describe how to construct the map χ described in the previous section. The first step is a redundant injective map.

Definition 3. *We call a map $\psi : A \to B$, with inverse $\psi^{-1} : B \to A \cup \{\perp\}$ a* redundant injective map *with security parameter ℓ_H if it satisfies the following properties:*

Computable *The functions ψ and ψ^{-1} are efficiently computable.*
Reversible *For all $a \in A$ we have $\psi^{-1}(\psi(a)) = a$.*
Redundant *For any $b \in_R B$ we have $\psi^{-1}(b) = \perp$ with probability $1 - 2^{-\ell_H}$.*

The redundancy prevents the attack described at the end of Sect. 2, but requires $|B|$ to be at least $2^{\ell_H}|A|$. In our scheme, B must be a group. We therefore use the following group encoding that maps strings to group elements.

Definition 4. *A group encoding $(\phi, \phi^{-1}, \{0,1\}^\ell, \mathbf{G}, E)$ consists of a bijective function $\phi : \{0,1\}^\ell \to E \subset \mathbf{G}$ and its inverse ϕ^{-1}. The functions ϕ and ϕ^{-1}, and membership tests in E run in polynomial time, and $|\mathbf{G}|/|E|$ is polynomial in ℓ.*

The Elligator map [1] is one such encoding, where $|\mathbf{G}|/|E| \approx 2$, and the group is an elliptic curve.

Definition 5. *Our* Redundant Injective Map *consists of the three algorithms* RIM.GEN, RIM.MAP *and* RIM.UNMAP*—the latter two correspond to χ and χ^{-1}.*

RIM.GEN$(1^{\ell_p}, 1^{\ell_H}, (\phi, \phi^{-1}, \{0,1\}^{\ell_p+\ell_H}, \mathbf{G}, E))$ *Given a plaintext size ℓ_p, a security parameter ℓ_H, and a group encoding $(\phi, \phi^{-1}, \{0,1\}^{\ell_p+\ell_H}, \mathbf{G}, E)$, it outputs two cryptographic hash functions $H_1 : \{0,1\}^{\ell_p} \to \{0,1\}^{\ell_H}$ and $H_2 : \{0,1\}^{\ell_H} \to \{0,1\}^{\ell_p}$.*

RIM.MAP(p) *This function takes as input a plaintext $p \in \{0,1\}^{\ell_p}$ and returns the group element $\phi(p \oplus H_2(r) \parallel r) \in E \subset \mathbf{G}$ where $r = H_1(p)$.*

RIM.UNMAP(c) *Given a group element $c \in \mathbf{G}$ this function returns $\perp$ if $c \notin E$. Else, it sets $b_1 \parallel b_2 = \phi^{-1}(c)$ and $p = b_1 \oplus H_2(b_2)$. If $H_1(p) = b_2$ it returns p, else it returns $\perp$.*

Computability and reversibility are clearly satisfied. For any $c \in_R E$ the inverse $b_1 \parallel b_2 = \phi^{-1}(c)$ is uniformly distributed over $\{0,1\}^{\ell_p+\ell_H}$. Therefore, since H_1 is a hash-function, $H_1(b_1 \oplus H_2(b_2)) = b_2$ with probability $2^{-\ell_H}$, so the map is redundant.

We need the following lemma in our security proof.

Lemma 1. *Our Redundant Injective Map from Definition 5 is programmable in the random oracle model for H_1 and H_2. This means that we can adaptively ensure that* $\text{RIM.MAP}(p) = g$ *for any* $p \in \{0,1\}^{\ell_p}$ *and* $g \in_R E$ *with overwhelming probability, provided that H_1 was not queried with p.*

Proof. Suppose we wish to set $\text{RIM.MAP}(p) = b_1 \parallel b_2 = \phi^{-1}(g)$. We set $H_1(p) = b_2$. Then, since b_2 is random, with overwhelming probability it was not queried before and we can set $H_2(b_2) = p \oplus b_1$. Since b_1 and b_2 are random, so is $p \oplus b_1$, therefore, the outputs are set to random values as required. □

4 A New DE scheme

Our new DE scheme, which we sketched in Sect. 2, directly creates shares of the plaintext instead of shares of an identity-based decryption key that decrypts the plaintext, as in Hoepman and Galindo's scheme [7]. The resulting scheme is simpler and no longer requires pairings. Furthermore, the new structure allows us to define a non-trivial key-evolution method, which seems impossible for the original scheme without compromising forward security. We formally introduce this scheme now.

4.1 Syntax

First, we recall the syntax of a key-evolving distributed encryption scheme from [7]. Note that we have made the safety requirement explicit.

Definition 6 (Key-evolving Distributed Encryption). *A k-out-of-n* key-evolving distributed encryption *scheme with lifetime divided into s stages, or (k,n,s)-KDE scheme, consists of the following four algorithms.*

KDE.GEN$(1^\ell, k, n, s, \ell_p)$ *This key generation algorithm takes as input a security parameter 1^ℓ, a threshold k, the number of senders n, the number of stages s and a plaintext size ℓ_p.*[3] *For each sender it generates initial encryption keys $S_{1,1}, \ldots, S_{1,n}$ and returns these, the system parameters, and the plaintext space $\mathscr{P}$.*

KDE.UPDKEY$(S_{\sigma,i})$ *The key update function* KDE.UPDKEY *takes as input $S_{\sigma,i}$ and outputs the key $S_{\sigma+1,i}$ for the next stage. This function aborts if $\sigma + 1 > s$.*

KDE.ENC$(S_{\sigma,i}, p)$ *Given an encryption key $S_{\sigma,i}$ and a plaintext p, this function returns a ciphertext share c.*

KDE.COMB(C) *Given a set $C = \{c_1, \ldots, c_k\}$ consisting of k ciphertext shares, the function* KDE.COMB(C) *either returns a plaintext p or* ERROR.

Every key-evolving distributed encryption scheme must satisfy the following correctness and safety requirements.

CORRECTNESS *Create the encryption keys $S_{\sigma,1}, \ldots, S_{\sigma,n}$ by running the algorithm* KDE.GEN *and then repeatedly updating them using* KDE.UPDKEY *to reach the required stage σ. For all plaintexts p and pairwise disjoint senders i_j we have* $\text{KDE.COMB}(C) = p$ *if* $C = \{\text{KDE.ENC}(S_{\sigma,i_1}, p), \ldots, \text{KDE.ENC}(S_{\sigma,i_k}, p)\}$.

[3] In the original description the plaintext size was implicit.

SAFETY *Generate $S_{\sigma,1},\ldots,S_{\sigma,n}$ as for correctness. If $C = \{\text{KDE.ENC}(S_{\sigma,i_1},p_{i_1}),\ldots,\text{KDE.ENC}(S_{\sigma,i_k},p_{i_k})\}$ with not all p_i equal, then with overwhelming probability $\text{KDE.COMB}(C) = \text{ERROR}$.*

To make the system secure in practice, senders need to get their keys in a secure manner, and, to ensure forward security, senders have to destroy the old key after updating it.

A distributed encryption scheme is a special case of a key-evolving distributed encryption scheme.

Definition 7 (Distributed Encryption). *A k-out-of-n* distributed encryption *scheme, or (k,n)-DE scheme, is a $(k,n,1)$-KDE scheme were the functions are called* DE.GEN, DE.ENC *and* DE.COMB *instead.*

4.2 Security definition

We define the forward security of a key-evolving distributed encryption scheme by recalling its security game. We present the security game by Hoepman and Galindo [7] in a slightly more general setting: plaintexts that have been queried before may be used in the challenge phase, provided this does not lead to a trivial win for the adversary.

Definition 8 (KDE forward-security game). *Consider a (k,n,s)-*KDE *key-evolving distributed encryption scheme with security parameter 1^ℓ given by the four algorithms* KDE.GEN*,* KDE.UPDKEY*,* KDE.ENC *and* KDE.COMB*. Define the following game between a challenger and an adversary $\mathscr{A}$.*

Setup *The challenger runs $\text{KDE.GEN}(1^\ell,k,n,s)$ to obtain $(S_{1,1},\ldots,S_{1,n})$ and sends a description of the plaintext space $\mathscr{P}$ and system parameters to the adversary.*

Find *The challenger initializes the current stage σ to 1, and the set of corrupted senders $I_{1,c}$ to the empty set. The adversary can issue the following three types of queries:*

- *A $\mathsf{corrupt}(i)$ query is only allowed before any encryption query $\mathsf{enc}(i,p)$ has been made for the current stage. If the query is allowed, the challenger sends $S_{\sigma,i}$ to the adversary and it adds i to $I_{\sigma,c}$.*
- *On encryption queries $\mathsf{enc}(i,p)$, where $i \in [n]$, $i \notin I_{\sigma,c}$ and $p \in \mathscr{P}$, the adversary receives the ciphertext $\text{KDE.ENC}(S_{\sigma,i},p)$.*
- *On next-stage queries $\mathsf{next}()$, the challenger updates the encryption keys of senders $i \in \{1,\ldots,n\} \setminus I_{\sigma,c}$ by setting $S_{\sigma+1,i} \leftarrow \text{KDE.UPDKEY}(S_{\sigma,i})$. The adversary is responsible for updating the keys of the other senders $i \in I_{\sigma,c}$. Finally, the challenger sets $I_{\sigma+1,c} \leftarrow I_{\sigma,c}$ and $\sigma \leftarrow \sigma+1$.*

Challenge *The adversary $\mathscr{A}$ outputs a challenge stage number $\sigma^\star < s$, indices $I_{nc} = \{i_1,\ldots,i_t\}$ corresponding to senders from which it wants to receive challenge ciphertexts and two equal length plaintexts $p_0,p_1 \in \mathscr{P}$. Let r denote the cardinality of $I_{\sigma^\star,c}$ and C_0 and C_1 denote the senders at which plaintexts p_0 and p_1 were queried respectively in stage $\sigma^\star$. The challenger aborts if the challenge is not* valid*, i.e., if one of the following conditions holds*

- *p_0 or p_1 was queried at a challenge sender, i.e., if $(C_0 \cup C_1) \cap I_{nc} \neq \emptyset$;*
- *a challenge sender was already corrupted, i.e., if $I_{nc} \cap I_{\sigma^\star,c} \neq \emptyset$; or*

– *too many shares are known to the adversary for either p_0 or p_1. This is the case if* $\max(|C_0 \cup I_{\sigma^\star,c}|, |C_1 \cup I_{\sigma^\star,c}|) + |I_{nc}| \geq k$.

Finally, the challenger chooses $\beta \in_R \{0,1\}$ and returns a challenge ciphertext share $c_{\sigma*,i} = \text{DE.ENC}(E_{\sigma*,i}, p_\beta)$ for each $i \in I_{nc}$.

Guess *The adversary $\mathscr{A}$ outputs a guess $\beta' \in \{0,1\}$. The adversary wins if $\beta = \beta'$. The advantage of adversary $\mathscr{A}$ is given by $\mathbf{Adv}^{\text{KDE}}_{\mathscr{A}}(1^\ell) = 2\left|\Pr[\beta' = \beta] - 1/2\right|$. An* KDE *scheme is called* forward secure *if $\mathbf{Adv}^{\text{KDE}}_{\mathscr{A}}(1^\ell)$ is negligible for every PPT adversary $\mathscr{A}$.*

The solution by Hoepman and Galindo is only secure in a more restricted attacker model (which they call the static adversary model), where the attacker announces up front which senders it will corrupt in what stage, what the challenge stage $\sigma^\star$ is, and which senders it will query in this challenge phase. Our scheme does not need these restrictions.

4.3 A New Distributed Encryption Scheme

In Sect. 2 we gave a sketch of our new distributed encryption scheme. Here we fill out the details. Correctness and safety follow from the earlier discussion.

Definition 9 (DE scheme). *The new distributed encryption (DE) scheme is given by the following algorithms, where* RIM.GEN, RIM.MAP *and* RIM.UNMAP *are as in Def. 5.*

DE.GEN$(1^\ell, k, n, \ell_p)$ *Create a group* $\mathbf{G}$ *of prime order q, where $q \approx 2^\ell$, such that DDH is hard in* $\mathbf{G}$*, and create a group encoding $\delta = (\phi, \phi^{-1}, \{0,1\}^{\ell_p+\ell}, \mathbf{G}, E)$. Call* RIM.GEN$(1^{\ell_p}, 1^\ell, \delta)$ *to setup the redundant injective map. Let $\mathscr{P} = \{0,1\}^{\ell_p}$ be the plaintext space. Share the public value* 1 *using Shamir's k-out-of-n secret sharing as follows. Choose $\varepsilon_1, \ldots, \varepsilon_{k-1} \in_R \mathbf{Z}_q$ and define the $k-1$ degree polynomial $f(x) = 1 + \sum_{i=1}^{k-1} \varepsilon_i x^i$, then $f(0) = 1$. Every sender i is given a share $S_i = (i, s_i)$ where $s_i = f(i)$. Output S_i for each sender, and publish $(\mathbf{G}, q, E, \phi, \phi^{-1}, \mathscr{P})$.*

DE.ENC(S_i, p) *Given an encryption key $S_i = (i, s_i)$ let $\alpha_i = \text{RIM.MAP}(p)^{s_i}$. Return $c_i = (i, \alpha_i)$.*

DE.COMB(C) *Let $C = \{c_{i_1}, \ldots, c_{i_k}\}$. Each share c_{i_j} is parsed as (i_j, α_{i_j}). Construct*[4] *$I = \{i_1, \ldots, i_k\}$, let $c = \prod_{i \in I} (\alpha_i)^{\lambda_i^I}$ and return* RIM.UNMAP(c).

In Sect. 7.1 we sketch how to handle arbitrary-length plaintexts.

4.4 Security of the DE scheme

In this section we sketch the proof of the following theorem.

Theorem 1. *In the random oracle model for H_1 and H_2 the new distributed encryption scheme from Def. 9 is secure assuming the DDH assumption holds in the group* $\mathbf{G}$.

We first give an ideal model for this scheme, and show that in this model the DE scheme is secure. We then prove that the ideal model and the actual scheme are indistinguishable, hence proving the security of the actual scheme as well.

[4] The index i_j is part of c_j to be able to explicitly reconstruct I and thus compute $\lambda_{i_j}^I$ given a set C.

The ideal scheme. In the ideal scheme the secret sharing is made specific to the plaintext. So, sender i uses a plaintext-specific secret share $s_i^{(p)}$ to construct a ciphertext share $\alpha_i = \text{RIM.MAP}(p)^{s_i^{(p)}}$ for plaintext p. For each p, the secret shares $s_1^{(p)}, \ldots, s_n^{(p)}$ form a random k-out-of-n sharing of the secret 1.

The reader may wonder at this point, if it still suffices to use a degree $k-1$ polynomial. Traditional uses of a secret sharing scheme suggest that the degree should be k instead, since one secret share is already known. This is not the case, for the following two reasons. First, knowing the secret itself does not help in the recovery as the generator, i.e., the encryption of the plaintext, is actually unknown. Second, while it is possible to guess the generator, thus giving k shares in total, an extra share is needed to verify that guess.

Lemma 2. *The ideal DE scheme is secure.*

Proof. By construction of the secret shares we only need to consider the shares for the challenge plaintexts p_0 and p_1; all others are completely independent. After the challenge the attacker knows at most $k-1$ ciphertext shares. Hence, no information is leaked as with only $k-1$ shares, both sets of shares are equally likely to combine to $\text{RIM.MAP}(p_0)$ as they are to $\text{RIM.MAP}(p_1)$. In fact, for each set of shares there exists a kth share that reconstructs the desired value. □

Indistinguishability of ideal and real scheme. Suppose an attacker can break DDH, i.e., given $(g, A = g^a, B = g^b, C = g^c)$ it can decide whether $c = ab$. Then it can break our scheme as follows. It picks three different plaintexts p, p_0, p_1, and calculates $g = \text{RIM.MAP}(p)$ and $A = \text{RIM.MAP}(p)^{s_i}$, for the latter it uses one query. Then it sets $B = \text{RIM.MAP}(p_0)$ and obtains $C = \text{RIM.MAP}(p_d)^{s_i}$ as a response to its challenge query on p_0 and p_1. Now, $d = 0$ if and only if $c = ab$ in the DDH problem, thus breaking the DE security as well.

The indistinguishability proof that we present here shows that any attacker has to solve a DDH problem. The proof is in the hybrid model, see Fig. 1. Queries for the first $\kappa - 1$ plaintexts will be answered using ideal shares, then the κth plaintext will get either ideal or real shares depending on whether $c = ab$ in the DDH instance, and the remaining plaintext will have real shares. We use Lem. 1 to ensure that the correct generators are used for those plaintexts. Induction on κ shows that any distinguisher between ideal and real shares thus solves the DDH instance.

To construct the shares corresponding to the different senders while still ensuring proper secret-sharing we duplicate the DDH instance, combine it with the corrupted shares, and derive the remaining shares based on the underlying secret sharing scheme.

Lemma 3. *In the random oracle model for H_1 and H_2 (as used by the redundant injective mapping), the ideal and the real DE schemes are indistinguishable provided the DDH assumption holds in* **G**.

Proof. For this proof we use a hybrid scheme that is parametrized by κ. Let $(g, A = g^a, B = g^b, C = g^c)$ be a DDH instance for **G**, our task is to decide whether $c = ab$.

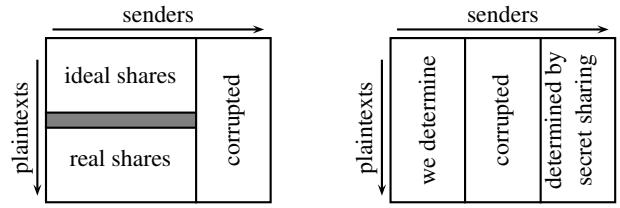

Fig. 1. The table on the left shows which type of secret shares are used to answer the queries for the plaintexts while the table on the right shows how these answers are constructed.

This proof is in the random oracle model for H_1 and H_2, so the adversary has oracle access to each of them. Let $\mathscr{P}_Q = (p_1, \ldots, p_{q_E})$ be the plaintexts queries made by the adversary to the H_1 oracle.

In the hybrid scheme, the ciphertext shares corresponding to plaintexts in the set $\mathscr{P}_I = \{p_1, \ldots, p_{\kappa-1}\}$ will be created using ideal shares, whereas the ciphertext shares for plaintexts in $\mathscr{P}_R = \{p_{\kappa+1}, \ldots, p_{q_E}\}$ will be created using the real shares, see Fig. 1. If $c = ab$ (of the DDH instance) then the hybrid scheme uses real shares for p_κ and ideal shares otherwise. Any distinguisher for the two variants will thus solve the DDH instance. Induction over κ completes the proof.

We now show how to play the security game. Initially we generate $\gamma_p \in_R \mathbf{Z}_q$ for all $p \in \mathscr{P}_R$ such that $g^{\gamma_p} \in E$ and we generate $\gamma_{p_\kappa} \in_R \mathbf{Z}_q$ such that $A^{\gamma_{p_\kappa}} \in E$. This is possible since $|\mathbf{G}| / |E|$ is polynomial and can be done without knowing the queries in advance. As queries for p's come in to the H_1 oracle, use Lemma 1 to set $\mathrm{RIM.MAP}(p) = g^{\gamma_p}$ for $p \in \mathscr{P}_R$ and to set $\mathrm{RIM.MAP}(p_\kappa) = A^{\gamma_{p_\kappa}}$. All other queries are answered normally.

Then we play the game as follows. The attacker only makes corruption queries at the start of the game. For every $\mathsf{corrupt}(i)$ query, add i to I_c and generate an arbitrary secret-share $s_i \in_R \mathbf{Z}_q$ and send $S_i = (i, s_i)$ to the challenger.

We now consider three cases of $\mathsf{enc}(i, p)$ queries. The first, where $p \in \mathscr{P}_I$, for which the answers will be using ideal shares, the second when $p = p_\kappa$ and the third when $p \in \mathscr{P}_R$, for which the answers will be using real shares.

Without loss of generality, we assume that the r corrupted senders are numbered $k-r, \ldots, k-1$, see Fig. 1. As we cannot play with the corrupted senders, the corresponding shares are always given by $\mathsf{enc}(i, p) = \mathrm{RIM.MAP}(p)^{s_i}$ for $i > n - r$ and all plaintexts $p \in \mathscr{P}$.[5] This determines r shares. Furthermore, $\mathrm{RIM.MAP}(p)^1$ is also a valid share, giving $r+1$ determined shares. In the following we show how to answer the $\mathsf{enc}(i, p)$ queries with $1 \le i \le k - (r+1)$ for all three cases.

For plaintexts $p \in \mathscr{P}_I$ generate ideal secret shares $s_i^{(p)}$ for senders $1 \le i \le k-(r+1)$. The ciphertext shares are given by $\mathsf{enc}(i, p) = \mathrm{RIM.MAP}(p)^{s_i^{(p)}}$.

For the remaining plaintexts we use the DDH instance $(g, A = g^a, B = g^b, C = g^c)$ to compute the answers to the $\mathsf{enc}(i, p)$ queries. First, create an extension as follows. Generate $d_i, e_i \in_R \mathbf{Z}_q$ for $1 \le i \le k - (r+1)$ and set $B_i = B^{d_i} g^{e_i}$ and $C_i = C^{d_i} A^{e_i}$. It can be shown that $(g, A, B_i = g^{b_i}, C_i = g^{c_i})$ are DDH tuples such that $c_i = ab_i$ when

[5] In this proof we omit the sender's index and just write $\mathsf{enc}(i, p) = \mathrm{RIM.MAP}(p)^{s_i}$.

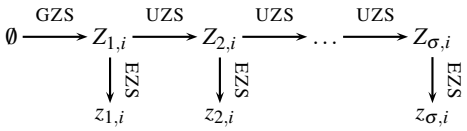

Fig. 2. Graphical representation of an evolving zero-sharing scheme.

$c = ab$ in the original problem, and $c_i \in_R \mathbf{Z}_q$ otherwise [10]. We then act as if $s_i = b_i$ for $1 \leq i \leq k-(r+1)$.

For p_κ the ciphertext shares for $1 \leq i \leq k-(r+1)$ are given by $\mathsf{enc}(i, p_\kappa) = C_i^{\gamma_{p_\kappa}}$. If $c_i = ab_i$ then we have $\mathsf{enc}(i, p_\kappa) = (g^{a\gamma_{p_\kappa}})^{b_i} = \text{RIM.MAP}(p_\kappa)^{s_i}$, making the shares real. Otherwise, the c_i's are random, thus the shares are ideal.

For all other plaintexts $p \in \mathscr{P}_R$ the ciphertexts for $1 \leq i \leq k-(r+1)$ are given by $c_{pi} = B_i^{\gamma_p} = (g^{\gamma_p})^{b_i} = \text{RIM.MAP}(p)^{s_i}$, as desired.

We now determined k shares for every plaintext p, the responses for senders $k, \ldots, n$, see Fig. 1, are calculated from these by interpolating the exponents

$$\mathsf{enc}(i,p) = \text{RIM.MAP}(p)^{1\lambda_0^I(i)}\mathsf{enc}(1,p)^{\lambda_1^I(i)} \cdots \mathsf{enc}(k-1,p)^{\lambda_{k-1}^I(i)}$$

We have now described how to answer the queries.

Since DDH is hard, two subsequent hybrid schemes are indistinguishable. Thus, the ideal scheme and the real scheme are indistinguishable as well. □

5 Forward-Secure DE scheme

The keys of our DE scheme consist of Shamir secret shares, to create a forward secure scheme we need to forward-securely evolve these shares. In this section we show how to do this. When we combine this technique with our new distributed encryption schemes the result is forward secure. In this section we prove this for the scheme in Sect. 4. In the next section we prove this for the batched scheme.

5.1 A key-evolution scheme

The scheme we present in this section forward securely generates sharings of the value 0. It combines ideas from Cramer *et al.* [4] and Ohkubo *et al.* [11]. We take the following approach, see also Fig. 2. Time is split into stages. Every sender i has an internal state $Z_{\sigma,i}$ for the current stage σ. The states of all senders combined implicitly define a zero-sharing polynomial z_σ. The states are constructed in such a way that every sender can, without interacting with other senders, derive its zero-share $z_{\sigma,i} = z_\sigma(i)$ for that stage. To move to the next stage, every party can individually update the internal state. After destroying the previous internal state it is not possible to retrieve any information on it from the current internal state.

Syntax. The informal description of the scheme captured in the previous section can be formalized as follows.

internal	$Z_{\sigma,1}$	$\dots$	$Z_{\sigma,r}$	$Z_{\sigma,r+1}$	$\dots$	$Z_{\sigma,n}$
external	$\overline{z_{\sigma,1}}$	$\dots$	$\overline{z_{\sigma,r}}$	$\overline{z_{\sigma,r+1}}$	$\dots$	$\overline{z_{\sigma,n}}$

Fig. 3. The highlighted section illustrate the adversary's view for a set of corrupted senders $I_c = \{1,\dots,r\}$. In the next stage, $\sigma+1$, the adversary gets the complete internal state.

Definition 10 (Evolving zero-sharing). *The next three algorithms describe an* evolving zero-sharing *scheme. See also Fig. 2.*

- $\text{GZS}(k,n,s,\mathscr{K})$ *takes as input the threshold k, the number of senders n, the number of stages s, and secret sharing field* $\mathscr{K}$*. It outputs initial states* $Z_{1,1},\dots,Z_{1,n}$*.*
- $\text{UZS}(Z_{\sigma,i})$ *is a non-interactive protocol that takes as input the current state* $Z_{\sigma,i}$ *and outputs a new state* $Z_{\sigma+1,i}$ *or aborts.*
- $\text{EZS}(Z_{\sigma,i})$ *takes as input the current state* $Z_{\sigma,i}$ *and outputs a zero share* $z_{\sigma,i}$*.*

This definition describes a non-interactive scheme because our use cases require this. Interactive schemes are easier to build, but are not considered in this paper.

Intuitively, forward security requires that no matter what an adversary learns in later stages, it cannot use this knowledge to obtain additional information on the current stage. We formalize this notion for evolving zero-sharing schemes, which we call transparency, as we need it to prove forward-security of our key-evolving distributed encryption scheme.

Consider a stage σ. Clearly, an adversary has the biggest advantage in learning more about stage σ, if it gets the complete state of the system in stage $\sigma+1$. The following definition formalizes the notion that every zero-sharing polynomial in stage σ is equally likely, as long as it matches the view the adversary already had obtained through corruptions—note this fixes the polynomial if the adversary has corrupted $k-1$ senders. The adversary gets access to the full zero-sharing polynomial in stage σ. We note that this is very liberal, as in the actual combination with the DE schemes the zero-shares will be kept secret.

Definition 11 (Transparency). *Let k be the threshold, n the total number of senders, q the group order and s the maximum number of stages. Let an evolving zero-sharing given by the algorithms* GZS, UZS *and* EZS *be defined for these parameters. Let* $Z_{\sigma,1}$, $\dots,Z_{\sigma,n}$ *be the result of calling* GZS *and then running* UZS $\sigma-1$ *times for each sender. Furthermore, let* $z_{\sigma,i} = \text{EZS}(Z_{\sigma,i})$ *and* $Z_{\sigma+1,i} = \text{UZS}(Z_{\sigma,i})$*. Let* $\mathscr{A}$ *be an adversary. It outputs a set* $I_c \subset [n]$ *of senders corrupted in stage* σ *and receives*

- *the internal state* $Z_{\sigma,i}$*, for all senders* $i \in I_c$*,*
- *the external state* $\overline{z_{\sigma,i}}$ *for all senders, and*
- *the internal state* $Z_{\sigma+1,i}$ *for all senders,*

see also Fig. 3. We say the evolving zero sharing scheme is transparent *if adversary* $\mathscr{A}$ *cannot distinguish the following two situations:*

1. *the normal situation with* $\overline{z_{\sigma,i}} = z_{\sigma,i}$ *and*
2. *a situation in which the secret changes, i.e.* $\overline{z_{\sigma,i}} = z_{\sigma,i} + z(i)$ *where z is a degree* $k-1$ *zero-sharing polynomial, such that* $z(i) = 0$ *for all* $i \in I_c$*.*

Key-evolution scheme. We follow Cramer *et al.* [4] in constructing a zero-sharing polynomial in such a way that sender i can only evaluate the polynomial at the point i. For every set $A \subset [n]$ of cardinality $n-(k-2)$ we define the $k-1$ degree polynomial $g_A(x) = x\prod_{i\in[n]\setminus A}(x-i)$. Our zero-sharing polynomial is then given by

$$z_\sigma(x) = \sum_{\substack{A\subset[n]\\ |A|=n-(k-2)}} r_{\sigma,A}\cdot g_A(x),$$

where a factor $r_{\sigma,A}$ is known only to the senders $i \in A$. Note that by construction, z is of degree $k-1$ and $z_\sigma(0)$ is indeed 0. It can be shown that $k-2$ colluding parties cannot recover $z_\sigma(x)$. Furthermore, for every zero-sharing polynomial z of degree $k-1$, there exist values for the $r_{\sigma,A}$s such that $z_\sigma(x) = z(x)$. This gives the following scheme.

Definition 12 (Evolving zero-sharing scheme). *Let ℓ_h be a security parameter and let $h_1 : \{0,1\}^{\ell_h} \to \{0,1\}^{\ell_h}$ and $h_2 : \{0,1\}^{\ell_h} \to \mathbf{Z}_q$ be hash functions. The evolving zero-sharing scheme is implemented as follows.*

- GZS$(k,n,s,\mathbf{Z}_q)$ *For each $A \subset [n]$ of cardinality $n-(k-2)$ generate a random share $\bar{r}_{1,A} \in_R \{0,1\}^{\ell_h}$ and for each sender i set $Z_{1,i} = (\bar{r}_{1,A})_{A\ni i}$.*
- UZS$(Z_{\sigma,i})$ *This algorithm is non-interactive. First, parse $Z_{\sigma,i}$ as $(\bar{r}_{\sigma,A})_{A\ni i}$, and set $\bar{r}_{\sigma+1,A} = h_1(\bar{r}_{\sigma,A})$ for A such that $i \in A$. Then return $Z_{\sigma+1,i} = (\bar{r}_{\sigma+1,A})_{A\ni i}$.*
- EZS$(Z_{\sigma,i})$ *To derive the zero-share parse $Z_{\sigma,i}$ as $(\bar{r}_{\sigma,A})_{A\ni i}$. Then, set $r_{\sigma,A} = h_2(\bar{r}_{\sigma,A})$ for A such that $i \in A$ and determine*

$$z_{\sigma,i} = z_\sigma(i) = \sum_{\substack{A\subset[n]\\ |A|=n-(k-2)}} r_{\sigma,A}\cdot g_A(i).$$

Finally, return $z_{\sigma,i}$.

The construction and the proof of the following lemma are inspired by the Ohkubo *et al.* scheme [11]. See the appendix for the proof.

Lemma 4. *The evolving zero-sharing scheme from Def. 12 is transparent in the random oracle model for h_2.*[6]

5.2 A Key-Evolving DE Scheme

In this section we build a key-evolving DE scheme using the evolving zero-sharing scheme of the previous section. The latter scheme generates as many distributed zero-sharing polynomials as we want. By adding the constant polynomial $g(x) = 1$ to this polynomial we obtain the key-sharing polynomial in our DE scheme.

Definition 13 (KDE scheme). *The key-evolving distributed encryption (KDE) scheme is constructed from the new distributed encryption scheme given by the algorithms* DE.GEN, DE.ENC *and* DE.COMB, *and the evolving zero-sharing scheme from Definition 12 given by the algorithms* GZS, UZS *and* EZS. *It is defined by the following four algorithms.*

[6] Actually, it is not really necessary to use a random oracle for this part of the proof. In fact, one can use k-wise independent functions, see for example Canetti *et al.* [3], and thus prove this lemma in the standard model.

- KDE.GEN$(1^\ell, k, n, s, \ell_p)$ *The* KDE.GEN *algorithm first runs* DE.GEN$(1^\ell, k, n, \ell_p)$ *to obtain* $(\mathbf{G}, q, E, \phi, \phi^{-1}, \mathscr{P})$*, which it outputs as well. Here group* $\mathbf{G}$ *is of order* q*. It then calls* GZS$(k, n, s, \mathbf{Z}_q)$ *to obtain* $Z_{1,1}, \ldots, Z_{1,n}$*, sets* $s_{1,i} = 1 + \text{EZS}(Z_{1,i})$ *and outputs* $S_{1,i} = (i, s_{1,i}, Z_{1,i})$ *for each sender.*
- KDE.UPDKEY$(S_{\sigma,i})$ *Let* $S_{\sigma,i} = (i, s_{\sigma,i}, Z_{\sigma,i})$*. It then sets* $Z_{\sigma+1,i} = \text{UZS}(Z_{\sigma,i})$ *and* $s_{\sigma+1,i} = 1 + \text{EZS}(Z_{\sigma+1,i})$ *and returns* $S_{\sigma+1,i} = (i, s_{\sigma+1,i}, Z_{\sigma+1,i})$ *or aborts if* UZS *aborts.*
- KDE.ENC$(S_{\sigma,i}, p)$*. Let* $S_{\sigma,i} = (i, s_{\sigma,i}, Z_{\sigma,i})$*. To encrypt a plaintext* p *algorithm* KDE.ENC *returns the result of* DE.ENC$((i, s_{\sigma,i}), p)$*.*
- KDE.COMB(C) *To combine ciphertexts* KDE.COMB *runs* DE.COMB(C)*.*

Efficiency. The evolving zero-sharing scheme has complexity $\binom{n}{k-2}$ in both space and time to store and evaluate the zero-shares. While this number is exponential, it is almost always much smaller than the cost of combining in a real scenario (see Table 1 on page 17). In particular, it is comparable to recovering a single plaintext in the batched scheme which we will present in the next section. Furthermore, its space complexity is independent of the number of stages, which is a big gain with respect to the original scheme [7] where the space complexity is linear in the number of stages. In many practical applications this number will be a lot bigger than $\binom{n}{k-2}$.

Security. The security of the KDE scheme can easily be reduced to that of the DE scheme by using the properties of the evolving zero-sharing scheme. We give a sketch of the proof; we refer to App. B for the full proof.

Theorem 2. *The new KDE scheme is* (k, n, s)*-KDE secure provided that the DE is* $(k, n, 1)$*-KDE secure and the evolving zero-sharing scheme is transparent. The proof is in the random oracle model for* h_2[7]*.*

Proof (Sketch). We reduce the security of the KDE scheme to that of the DE scheme. To setup the system we generate random $\bar{r}_{1,A}$s. We guess the challenge phase $\sigma^\star$ and simulate all stages except $\sigma^\star$, where we use our DE oracle. To ensure that this is not detected we must ensure that corrupted hosts have the correct secret shares. We do this by modifying $h_2(\bar{r}_{\sigma^\star,A})$ in the random oracle model on $\bar{r}_{\sigma^\star,A}$ that were not yet known to the adversary. Then queries in stage $\sigma^\star$ can be answered by our DE oracle. The distribution of the secret shares does not correspond to the initial $\bar{r}_{1,A}$s, however, the transparency of the evolving zero-sharing scheme ensures this cannot be detected. □

This proof easily translates to other evolving zero-sharing schemes where the challenge phase can be incorporated.

6 Efficient solutions for small domains

As was already analyzed by Hoepman and Galindo [7], any distributed encryption solution will be rather inefficient. The main culprit is the combination phase. We do not

[7] Again, it is not really necessary to use a random oracle for this proof either. However, the security of the DE scheme uses non-standard assumptions anyway.

add indicators, like the hash of a plaintext, to the ciphertexts as they allow an attacker to trivially test if they belong to given plaintext. Therefore, it is not clear which shares might belong to the same plaintext. The only solution is to try all combinations of k shares, from different senders, of the received shares. Depending on the situation this can become prohibitive. We now propose a variant of our new scheme that is much more efficient for small plaintext domains.

The crucial difference is that we will now operate in a batched setting. At the end of a stage the sender generates a share for *every* plaintext. It generates a proper share for every plaintext it needs to send, as before, and a random value for all other plaintexts. Now we know directly which shares belong to a given plaintext. This reduces the exponential term in the combing phase considerably. Also, since the plaintext is known a priori, the only remaining task of the combiner is to determine whether this plaintext was encrypted by a sufficient number of senders. In particular we can replace the integrity preserving encryption scheme with a hash function.

6.1 Syntax

For small plaintext spaces, the following definition of a *batched key-evolving distributed encryption* scheme makes sense.

Definition 14 (Batched KDE**).** *A k-out-of-n* batched key-evolving distributed encryption *scheme with lifetime divided into s stages, or (k,n,s)-BKDE scheme, consists of the following four algorithms.*

BKDE.GEN$(1^\ell, k, n, s, \mathscr{P})$ *Given the security parameter 1^ℓ, the threshold k, the number of senders n, the number of stages s and a plaintext space $\mathscr{P}$ it generates initial encryption keys $S_{1,1}, \ldots, S_{1,n}$ for each sender $i \in [n]$. It returns these encryption keys as well as the system parameters.*

BKDE.ENC$(S_{\sigma,i}, P)$ *Given an encryption key $S_{\sigma,i}$ corresponding to sender i at stage σ and a set of plaintexts $P \subset \mathscr{P}$, this function returns a vector C_i of ciphertext shares of length $|\mathscr{P}|$.*

BKDE.UPDKEY$(S_{\sigma,i})$ *The key update function takes as input $S_{\sigma,i}$ and outputs the key $S_{\sigma+1,i}$ for the next stage. This function aborts if $\sigma + 1 > s$.*

BKDE.COMB$(C_1, \ldots, C_n, s)$ *Given the ciphertext share vectors $C_1, \ldots, C_n$ produced by the senders, the function* BKDE.COMB *returns a set of plaintexts P.*

A key-evolving batched distributed encryption scheme must satisfy the following combined correctness and safety requirement.

CORRECTNESS & SAFETY *Let the encryption keys $S_{\sigma,i}$ be generated as described above. Let $C_i =$* BKDE.ENC$(S_{\sigma,i}, P_i)$ *for each sender i and for all $P_i \subset \mathscr{P}$. Then the result P of* BKDE.COMB$(C_1, \ldots, C_n)$ *is such that $p \in P$ if $p \in P_i$ for at least k different P_i.*

6.2 Security Definition

The following game captures the security properties of our protocol.

Definition 15 (Batched KDE forward-security game). *Consider a* (k,n,s)-BKDE *batched key-evolving distributed encryption scheme. The batched KDE forward-security*

game is very similar to the KDE forward-security game for a (k,n,s)-KDE *scheme. We only note the changes. The algorithms* BKDE.GEN *and* BKDE.UPDKEY *replace the algorithms* KDE.GEN *and* KDE.UPDKEY.

Setup *First, the adversary outputs a plaintext space* $\mathscr{P}$ *it wants to attack, then the setup phase runs as before.*

Find *In the find phase the adversary is allowed to make* $\mathsf{bcorrupt}(i)$, $\mathsf{bnext}()$ *and* $\mathsf{benc}(i,P)$ *queries. The first two are implemented using* $\mathsf{corrupt}(i)$ *and* $\mathsf{next}()$ *respectively. On input of a query* $\mathsf{benc}(i,P_i)$*, where* $i \in [n]$*,* $i \notin I_{\sigma,c}$ *and* $P \subset \mathscr{P}$*, the challenger sends the vector* $\text{BKDE.ENC}(S_{\sigma,i},P_i)$ *to the adversary.*

Challenge *If the challenge on* p_0 *and* p_1 *at hosts* I_{nc} *is valid (see Definition 8) the challenger chooses* $\beta \in_R \{0,1\}$*, sets* $C = \text{BKDE.ENC}(S_{\sigma^\star,j},\{p_\beta\})$*, and returns the ciphertext share* C_{p_β} *to the challenger for each* $j \in I_{nc}$*.*

Guess *The guess phase is unchanged.*

The adversary $\mathscr{A}$*'s advantage is defined as* $\mathbf{Adv}^{\text{BKDE}}_{\mathscr{A}}(1^\ell) = \left|\Pr[\beta' = \beta] - 1/2\right|$*. An* BKDE *scheme is called* forward secure *if* $\mathbf{Adv}^{\text{BKDE}}_{\mathscr{A}}(1^\ell)$ *is negligible for every PPT adversary* $\mathscr{A}$*.*

6.3 The scheme

In the batched scheme a sender will output a complete vector of ciphertext shares $(c_{pi})_{p\in\mathscr{P}}$ at the end of a stage. Let $H : \{0,1\}^* \to \mathbf{G}$ be a cryptographic hash function. An element c_{pi} equals $H(p)^{s_i}$ when sender i been asked to encrypt p, and $c_{pi} \in_R \mathbf{G}$ otherwise. Intuitively, when the secret shares are unknown, these two are indistinguishable. The full scheme is given by the following definition. Note the similarities with our new KDE scheme.

Definition 16 (Batched KDE scheme). *Let* $(\text{GZS},\text{UZS},\text{EZS})$ *be an evolving zero-sharing scheme. The following algorithms define a batched key-evolving distributed encryption (BKDE) scheme.*

BKDE.GEN$(1^\ell,k,n,s,\mathscr{P})$ *Generate a cyclic group* $\mathbf{G}$ *such that its order* q *is of size* ℓ *bits. Then construct a hash function* $H : \{0,1\}^* \to \mathbf{G}$*. Create the secret sharing of zero by calling* $\text{GZS}(k,n,s,\mathbf{Z}_q)$ *to obtain* $Z_{1,1},\ldots,Z_{1,n}$*. Let* $s_{1,i} = 1 + \text{EZS}(Z_{1,i})$*. Output* $S_{1,i} = (s_{1,i},Z_{1,i})$ *for each sender, together with a description of* $\mathbf{G}$ *and the hash function* H*.*

BKDE.ENC$(S_{\sigma,i},P)$ *Let* $S_{\sigma,i} = (s_{\sigma,i},Z_{\sigma,i})$*, and let* $C_i = (c_{pi})_{p\in\mathscr{P}}$ *be the resulting ciphertext share vector such that for all* $p \in \mathscr{P}$

$$c_{pi} = \begin{cases} H(p)^{s_{\sigma,i}} & \text{if } p \in P \\ h \in_R \mathbf{G} & \text{otherwise.} \end{cases}$$

BKDE.UPDKEY$(S_{\sigma,i})$ *Let* $S_{\sigma,i} = (s_{\sigma,i},Z_{\sigma,i})$*. Set* $Z_{\sigma+1,i} = \text{UZS}(Z_{\sigma,i})$ *and* $s_{\sigma+1,i} = 1 + \text{EZS}(Z_{\sigma+1,i})$*. Return* $S_{\sigma+1,i} = (s_{\sigma+1,i},Z_{\sigma+1,i})$ *or abort if* UZS *does.*

BKDE.COMB$(C_1,\ldots,C_n)$ *For each* $p \in \mathscr{P}$*, do the following. Consider all shares* $(c_{p1},\ldots,c_{pn})$ *corresponding to plaintext* p *from senders 1 trough n. For all possible combinations of index sets* $I \subseteq \{1,\ldots,n\}$ *of size k verify whether*

$$\prod_{i\in I}(c_{pi})^{\lambda_i^I} = H(p)^1 = H(p).$$

If so, add p to the set of plaintexts to return.

The structure of this scheme is similar to that of our new DE and KDE schemes. Correctness and safety are easy to check. The security proofs of the DE and KDE schemes can readily be adapted to this setting directly by replacing the redundant injective map by one hash function in the random oracle model. We will not do this here. Instead we prove that the security of the BKDE scheme can be reduced to that of the KDE scheme. This theorem is slightly weaker as it requires us to also model H_1 and H_2 in the random oracle model. However, it nicely illustrates how the schemes relate.

Theorem 3. *The batched key-evolving distributed encryption scheme is (k,n,s)-BKDE secure, provided that the KDE scheme is (k,n,s)-KDE secure. The proof is in the random oracle model for H, H_1 and H_2.*

Proof. Suppose we have an adversary $\mathscr{A}$ against the batched scheme, then we build an adversary $\mathscr{B}$ against the KDE scheme. First, $\mathscr{A}$ requests a plaintext space $\mathscr{P}$ for the batched scheme. Then, $\mathscr{B}$ gets a description of the group and the plaintext space $\mathscr{P}'$ from its challenger. It forwards the group description to $\mathscr{A}$. Furthermore, it chooses a hash function $H' : \mathscr{P} \to \mathscr{P}'$ onto the plain text space required by the KDE scheme. Then we answer its queries as follows.

Adversary $\mathscr{B}$ answers a $\mathsf{bcorrupt}(i)$ query from $\mathscr{A}$ with the result of a $\mathsf{corrupt}(i)$ query to its oracle. On input of a $\mathsf{bnext}(i)$ query $\mathscr{B}$ makes a $\mathsf{next}()$ query to its oracle. Adversary $\mathscr{B}$ answers a hash query $H(x)$ with $H(x) = \mathrm{RIM.MAP}(H'(x))$.

The answer C_i to a batched encryption query $\mathsf{benc}(i, P_i)$ is constructed as follows:

$$c_{pi} = \begin{cases} \mathsf{enc}(i, H'(p)) & \text{if } p \in P_i \\ h \in_R \mathbf{G} & \text{otherwise} \end{cases}$$

By choice of $H(x)$ this is indeed correct, as

$$\mathsf{enc}(i, H'(p)) = \mathrm{RIM.MAP}(H'(p))^{s_{\sigma,i}} = H(p)^{s_{\sigma,i}},$$

as desired. The challenge made by $\mathscr{A}$ is forwarded to $\mathscr{B}$'s challenge oracle, and the results relayed back. The advantage of $\mathscr{A}$ against the BKDE scheme is the same as $\mathscr{B}$'s advantage against the KDE scheme. □

7 Analysis and conclusions

7.1 Practical considerations

We propose two small extensions that can improve the scheme in practice. Our scheme works only with fixed-length plaintexts. It is, however, possible to handle longer plaintexts as well. First, append a hash of the message to authenticate the message as a whole. Then, split the message into appropriately sized chunks and run the DE scheme for each of them. After recovering the multiple blocks, combine them and check the hash before outputting the plaintext. This procedural extension allows encrypting arbitrary length plaintexts.

The second improvement deterministically encrypts the message with the public key of the combiner before running the DE scheme itself. This ensures that only the combiner—which is still assumed to be honest—can successfully recover the encrypted plaintexts even if ciphertexts leak.

Table 1. Performance comparison between the KDE scheme and the batched KDE (BKDE) scheme. The time complexity gives the approximate number of combine actions needed. The space complexity gives the number of ciphertext shares stored. Here m is the average number of plaintexts encrypted by each sender.

		KDE		*BKDE*	
	Parameters	Time	Space	Time	Space
Formula		$\binom{n}{k}m^k$	nm	$\binom{n}{k}\lvert\mathscr{P}\rvert$	$n\lvert\mathscr{P}\rvert$
Speed limiting	$n=2, \lvert\mathscr{P}\rvert=10^7$ $k=2, m=600$	$1\times 360\cdot 10^3$	$2,250$	$1\times 10\cdot 10^6$	$50\cdot 10^6$
Canvas cutters	$n=8, \lvert\mathscr{P}\rvert=10^7$ $k=4, m=400$	$70\times 26\cdot 10^9$	$3,200$	$70\times 10\cdot 10^6$	$80\cdot 10^6$

7.2 Performance

Tab. 1 shows the two methods' time complexity, in terms of combine operations, and the space complexity, in terms of stored ciphertext shares. It also gives numbers for two specific use cases. In both, we assume the total number of vehicles is 10 million, like in the Netherlands. For the first, a speed-monitoring example, we choose the parameters to monitor a 20 kilometer stretch of highway—for simplicity, we assume there are no exits—with one ANPR system placed at the beginning, and one at the end. We set the epoch length to 10 minutes. Every minute, we start a parallel instance of the system. This setting guarantees that every car going at least $120/(9/60) \approx 133$ km/h will generate two shares in the same epoch, and is thus always caught, while cars going between 120 km/h and 133 km/h may be caught. Using 20 parallel instances, instead of 10, will lower this bound to 126 km/h.

Suppose that 600 cars pass the ANPR systems during an epoch. The regular KDE scheme is more efficient in this setting due to the relatively low number of shares. In fact, it can be optimized significantly, because the first station needs to create a share for only the newest epoch, instead of all parallel ones. This modification reduces the combining cost by another factor of 10. In this setting, our key-evolution schemes ensure that the senders do not need to store $60\cdot 24$ keys for every day the system is operational, instead they store only two.

The second use case comes from Hoepman and Galindo [7]. They describe a scenario where criminals, so-called canvas-cutters, frequently visit rest stops along a highway, cut open the canvas on lorries, and rob them. The criminals can typically be recognized by looking for cars that visit rest stops rather frequently. Suppose we monitor 8 rest stops, and consider a car suspicious if it visits at least 4 within a 4 hour period. Suppose 400 (different) cars visit each rest stop per period. Here, the BKDE scheme is clearly better. The exponential factor in the regular scheme quickly drives up the number of combines needed. In fact, this would be exacerbated if traffic increases. Nevertheless, these cases also illustrate that if storage is an issue, or fewer shares are expected, then it is better to use the non-batched scheme.

7.3 Conclusion

In this paper, we presented a new distributed encryption scheme that is simpler than previous solutions, and uses weaker assumptions. Furthermore, we described a key-evolving variant that offers proper key evolution and is therefore forward secure. Additionally, we demonstrated a batched variant of our new distributed encryption scheme that is much more efficient for small plaintext domains.

None of the known distributed encryption schemes offer semantic security; senders always produce the same ciphertext for a given plaintext. It would be very interesting to see solutions that use randomization to avoid this problem.

References

1. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: elliptic-curve points indistinguishable from uniform random strings. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM Conference on Computer and Communications Security. pp. 967–980. ACM (2013)
2. Camenisch, J., Hohenberger, S., Kohlweiss, M., Lysyanskaya, A., Meyerovich, M.: How to win the clonewars: efficient periodic n-times anonymous authentication. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) ACM Conference on Computer and Communications Security. pp. 201–210. ACM (2006)
3. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. J. Cryptology 20(3), 265–294 (2007)
4. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Kilian, J. (ed.) Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings. LNCS, vol. 3378, pp. 342–362. Springer (2005)
5. Franklin, M.K.: A survey of key evolving cryptosystems. International Journal of Security and Networks 1(1/2), 46–53 (2006)
6. Hoepman, J.H.: Revocable privacy. ENISA Quarterly Review 5(2) (Jun 2009)
7. Hoepman, J.H., Galindo, D.: Non-interactive distributed encryption: a new primitive for revocable privacy. In: Chen, Y., Vaidya, J. (eds.) Proceedings of the 10th annual ACM workshop on Privacy in the electronic society, WPES 2011, Chicago, IL, USA, October 17, 2011. pp. 81–92. ACM (2011)
8. Itkis, G.: Forward security – adaptive cryptography: Time evolution. In: Bidgoli, H. (ed.) Handbook of Information Security, pp. 927–944. John Wiley and Sons (2006)
9. Lueks, W., Everts, M.H., Hoepman, J.H.: Revocable privacy 2012 – use cases. Tech. Rep. 35627, TNO (2012)
10. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. J. ACM 51(2), 231–262 (2004)
11. Ohkubo, M., Suzuki, K., Kinoshita, S.: Efficient Hash-Chain Based RFID Privacy Protection Scheme. In: International Conference on Ubiquitous Computing – Ubicomp, Workshop Privacy: Current Status and Future Directions. Nottingham, England (September 2004)
12. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
13. Speed Check Services: SPECS3 network average speed check solutions. `http://www.speedcheck.co.uk/images/SCS_SPECS3_Brochure.pdf`, accessed: 27 January 2013
14. Stadler, M.: Cryptographic Protocols for Revocable Privacy. Ph.D. thesis, Swiss Federal Institute of Technology, Zürich (1996)

A Proof of Lemma 4

Proof. Let $Z_{\sigma,i} = (\bar{r}_{\sigma,A})_{A \ni i}$ and let $z_\sigma(x)$ be the zero-sharing polynomial in stage σ, corresponding to situation one. Now, we show how to change this to $z_\sigma(x) + z(x)$ by modifying the random oracle for h_2. Let $r = |I_c|$, and w.l.o.g. assume that $I_c = \{n-r+1, \ldots, n\}$. Since $z(i)$ is zero for all $i \in I_c$ the polynomial is fully determined by $c = k-1-r$ extra values. We need to ensure that

$$z_{\sigma,i} + z(i) = \sum_{\substack{A \subset [n] \\ |A| = n-(k-2)}} r_{\sigma,A} g_A(i) = \underbrace{\sum_{A \cap I_c \neq \emptyset} r_{\sigma,A} g_A(i)}_{\text{fixed}} + \underbrace{\sum_{A \cap I_c = \emptyset} r_{\sigma,A} g_A(i)}_{\text{not fixed}}.$$

The fixed part contains values that are known to the adversary, and hence cannot be changed. The not-fixed part can, however, be changed. Consider the sets:

$$A_i = [n] \setminus (\{1, \ldots, i-1, i+1, \ldots c\} \cup I_c)$$

for $i \in [c]$. These sets are such that only set A_i influences the value for sender i, i.e.

$$z_{\sigma,i} + z(i) = \left[\sum_{\substack{A \subset [n] \\ |A| = n-(k-2), \forall j : A \neq A_j}} r_{\sigma,A} g_A(i) \right] + r_{\sigma,A_i} g_{A_i}(i), \tag{1}$$

for $i \in [c]$. Let $r_{\sigma,A} = h_2(\bar{r}_{\sigma,A})$ for all $A \neq A_i$ as always. Then choose the r_{σ,A_i}s such that equation (1) holds for $i \in [c]$. Finally, set $h_2(\bar{r}_{\sigma,A_i})$ to the new value r_{σ,A_i}. This cannot be detected by the adversary due to the one-wayness of h_1, so the situations are indistinguishable. □

B Full proof of Theorem 2

Proof. Suppose we have an adversary $\mathscr{A}$ against the KDE scheme. We then build an adversary $\mathscr{B}$ against the underlying DE scheme. Adversary $\mathscr{B}$ receives the system parameters from the challenger and forwards them to $\mathscr{A}$. Next, adversary $\mathscr{B}$ makes a guess $\sigma^\star$ for the challenge stage and initializes the set of corrupted senders I_c to $\emptyset$.

Adversary $\mathscr{B}$ will fully simulate all stages, except stage $\sigma^\star$, where it will use its oracle to answer the queries. For all $A \subset [n]$, such that $|A| = n-(k-2)$ generate $\bar{r}_{1,A} \in_R \{0,1\}^{\ell_h}$.

We now look into the details of the evolving zero-sharing scheme. By generating $\bar{r}_{1,A}$'s we have completely fixed the system, but we still need to ensure that epoch $\sigma^\star$ can be answered using our oracles. To this end we will change the value of the hash function $h_2(\bar{r}_{\sigma^\star,A_i})$ for specific sets A_i belonging to corrupted parties i. These sets A_i will be chosen in such a way, that $\bar{r}_{\sigma^\star,A_i}$ is not known to any previously corrupted party.

We handle $\mathsf{corrupt}(i)$ queries in or before stage $\sigma^\star$ as follows. Let I_c be the set of senders that were corrupted earlier and $f_{\sigma^\star}$ the secret-sharing polynomial induced by the values $\bar{r}_{\sigma^\star,A}$. First, we corrupt sender i using our oracle to obtain its internal state

$(i, s_{\sigma^\star,i}) = \mathsf{corrupt}(i)$. We need to ensure that $f_{\sigma^\star}(i) = s_{\sigma^\star,i}$. Pick a set A_i of cardinality $n-(k-2)$ such that $I_c \cap A_i = \emptyset$ and $i \in A_i$. This is possible, since the constraints in the challenge phase require $|I_c \cup \{i\}| < k$, therefore, $|I_c|$ will be at most $k-2$. For all other sets $A \ni i$ obtain $r_{\sigma^\star,A} = h_2(\bar{r}_{\sigma^\star,A})$ as usual. Then choose $r_{\sigma^\star,A_i}$ such that:

$$s_{\sigma^\star,i} = f_{\sigma^\star}(i) = 1 + \sum_{\substack{A \subset [n] \\ |A| = n-k+1}} r_{\sigma^\star,A} \cdot g_A(i).$$

By the choice of the set A_i the coefficient $\bar{r}_{\sigma^\star,A_i}$ is not known to any corrupted host, hence we can use the random oracle model to ensure that $h_2(\bar{r}_{\sigma^\star,A_i}) = r_{\sigma^\star,A_i}$. With very high probability this will not be detected by the adversary as the $\bar{r}_{\sigma^\star,A_i}$s are random. Finally, we return $(i, (\bar{r}_{\sigma^\star,A})_{A \ni i})$ to the adversary.

Now $\mathscr{B}$ proceeds as follows. For all stages except $\sigma^\star$ it knows the complete state of the system, and can thus answer all $\mathscr{A}$'s queries. For epoch $\sigma^\star$, all corrupted hosts will, by construction of the hash-function, have the correct secret shares for this epoch. All other queries can be answered by the oracle.

The transparency of the evolving zero-sharing scheme ensures that it is not possible for the adversary to detect that we do something different in stage $\sigma^\star$[8].

In the challenge phase $\mathscr{A}$ will announce its challenge phase σ'. If $\sigma' \neq \sigma^\star$ then $\mathscr{B}$ aborts. Otherwise, $\mathscr{B}$ will pass the challenge from $\mathscr{A}$ on to its own oracle. Finally, $\mathscr{B}$ outputs whatever $\mathscr{A}$ outputs. Adversary $\mathscr{B}$ has the same advantage as $\mathscr{A}$ up to a factor $1/s$ for guessing the stage. This proves the result. □

[8] This proof only changes h_2 for corrupted hosts, whereas the proof of Lemma 4 (see Appendix A) changes h_2 for non-corrupted hosts, so they can indeed be combined.