IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 June 2023, accepted 18 June 2023, date of publication 21 June 2023, date of current version 26 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3288285

METHODS

False-Bottom Encryption: Deniable Encryption From Secret Sharing

SHAHZAD AHMAD[1], STEFAN RASS[1,2], (Member, IEEE), AND PETER SCHARTNER[2]
[1]LIT Secure and Correct Systems Laboratory, Johannes Kepler University Linz, 4040 Linz, Austria
[2]Institute for Artificial Intelligence and Cybersecurity, Alpen-Adria-Universität Klagenfurt, 9020 Klagenfurt, Austria

Corresponding author: Shahzad Ahmad (shahzad.ahmad@jku.at)

This work was supported in part by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

ABSTRACT We show how to implement a deniable encryption method from secret sharing. Unlike the related concept of honey encryption, which employs a preprocessing step in symmetric encryption to re-shape the distribution of a plaintext towards making the real plaintext indistinguishable from a ciphertext for a fake message, we can avoid both, computational intractability assumptions and preprocessing of the data. This accomplishes deniability against an attacker that can force decryptions, and it can brute-force break a ciphertext with sufficient computational power. Following the concept of plausible deniability, we herein have different decryption keys to open up distinct plaintexts from within the same ciphertext. For instance, a plaintext revealed from a ciphertext with a key which was shared by a victim under duress, will convince the attacker that it is real, while the actual secret remains unnoticed. False Bottom Encryption constructs a symmetric scheme (in the sense of using the same key to encrypt and decrypt) that shares the properties of both honey encryption and deniable encryption. We specifically formalize and differentiate ''deniable'' from ''plausibly deniable'' as a security feature, showing how plausible deniability falls back to (only) deniability, depending on the plaintext distribution. Our scheme is simple, lightweight to implement and efficient in terms of encryption and decryption, and is based on secret sharing. As such, we do not rely on computational intractability. We corroborate the construction by giving numeric examples and providing implementations of the method as a Jupyter notebook supplementary to this work.

INDEX TERMS Deniable encryption, honey encryption, plausible deniability, secret sharing.

I. INTRODUCTION

Generally speaking, a sender and receiver must first share information to communicate safely. In many cases, encryption and good password protection may be sufficient to safeguard your data. For instance, with AES, the sender and receiver share the same key in a symmetric encryption system. However, using the RSA technique, the sender and receiver of an asymmetric-key encryption technique exchange a public system parameter and the recipient's public key, with the public key delivery occurring through public key infrastructure. These general encryption techniques offer a security guarantee against eavesdropping attempts, but they fall short when faced with threats of coercion. Even if the attacker does not have access to the key, if it intercepts the ciphertext, it may be able to force both the sender and the receiver to decrypt the message.

Non-committing encryption [1] and deniable encryption [2] have been presented as solutions to this issue. Users can decrypt an existing ciphertext associated with a certain counterfeit message using these *two* different encryption algorithms. The first algorithm is called the sender's encryption algorithm to encrypt a message under a secret key sk. The second algorithm, known as the faking algorithm, is publicly known, and the sender uses this fake algorithm to produce fake messages. Getting the same ciphertext from two different algorithms is computationally cumbersome. In our work, we show how the actual message and the fake message ciphertexts can both be produced using a single algorithm that also is computationally efficient.

There are many circumstances under which plausible deniability may also be necessary. If your opponents cannot obtain

The associate editor coordinating the review of this manuscript and approving it for publication was Jun Wang.

your password, strong encryption can keep them out. However, if the threat model incorporates coercion, such as the prospect of a jail term or torture, you might give up and hand over the key to rescue yourself. As a result, the attacker would have access to the data, perhaps putting you at risk for later repercussions.

The idea of plausible deniability originates in politics and espionage and refers to one's capacity to downplay one's culpability for, or knowledge of specific facts or events. It may entail carrying out operations in a way that leaves no trace, especially changing systems around particular people, to enable them to honestly deny their knowledge of what took place. Destruction of evidence is another method that can be used to make a given action plausible to deny, but there are also positive use-cases as we will outline next.

We question whether it is conceivable to produce ciphertext that appears to be for certain claimed receivers but are actually for different receivers. Imagine that Alice wants to secretly send her friend John a message. If she encrypts and sends it to John, she could be asked by her mother who the message was for and command her to decrypt the message. Consequently, John might get a call from Alice's mother, asking him to confirm as well what it says. To prevent this from happening, Alice can encrypt the message using deniable encryption. Alice will first prepare a pair of texts. One is a trivial message for Bob, whereas the other is a simple covert message for John. John's text is jointly encrypted with Bob's text by Alice using a suitable encryption algorithm. Alice posts the ciphertext to a public channel and requests her friends to download the message. The only two people who can successfully decrypt the ciphertext are Bob and John, but they produce two different messages: the fake message and the real message. The term ''successful decryption'' refers to the fact that Bob and John are able to decrypt and receive useful messages from the sender. Alice can tell her mother that the ciphertext is for Bob and reveal the message that was transmitted to Bob when questioned. Considering that Bob only knows what he has received, he can also be a trustworthy witness. Even if Alice's mother thinks something is concealed in the ciphertext, she cannot determine which of Alice's friends is the true recipient. In this case, Alice does not need to help John because her mother would not be able to suspect John, unless she suspects all of Alice's friends.

II. RELATED WORK

The concept of honey encryption [3] is a generic construction to extend conventional encryption by making decryptions under the wrong key ''appear to be plausible''. This is accomplished by transforming the input plaintext towards obtaining a certain fixed distribution that matches the distribution of the decryption result under a different key. Consequently, if the (same) ciphertext is decrypted under the real or the fake key, the resulting plaintexts (one real, the other being fake) will have approximately the same distribution. The security of honey encryption relies on the probability of an attacker judging a plaintext to be legitimate can be calculated by the encrypting party at the time of encryption. The main difference to deniable encryption and to our scheme is that honey encryption does not insert a second plaintext into the ciphertext. As in conventional encryption, there still is only one plaintext inside the ciphertext, but false decryptions should become less recognizable. This approach aims at retaining security even against keys or plaintexts of low min-entropy, which can be efficient to guess. The main difference to deniable encryption and our scheme is thus in the attacker model: we assume (as does deniable encryption) that the attacker puts force on the plaintext owner to open the ciphertext, while honey encryption lets the attacker attempt decryptions under keys of its own (random) choice.

The construction of honey encryption makes use of distribution-transforming encoders that aim to shape the distribution of a random plaintext towards a desired and fixed target distribution. Our scheme can use such encoders as well, as a source of plausibly looking plaintexts to act as fakes. We will not make explicit use of such transformations, but mention them as a possible technical implementation of our assumption that fake plaintexts are producible with the same distribution as the real secret plaintexts.

Canetti et al. [1] initially developed the concept of deniable encryption. A deniable shared key scheme and a public key scheme are two types of deniable encryption. A straightforward illustration of deniable encryption is the *one-time pad*: Let m be the original message to be encrypted, and c be the ciphertext such that $c = m \bigoplus k$ where k represents the shared key. Nobody can refute the encryptor's assertion that the message is m' using the key $k' = m' \bigoplus c$. In Canetti et al. [2] technique, falsified messages with strong justification were presented using the idea of a translucent set: roughly speaking, this is a set whose membership is not decidable efficiently without a trapdoor information. Encryption of a bit b is done by emitting a random string if $b = 0$ or a string from the translucent set T if $b = 1$. Under duress, the plaintext bit is deniable, since the claim of having taken a random string or one from T is not efficiently verifiable without the trapdoor information to decide its membership in T. According to Canetti et al., this system is sender-deniable, meaning that the sender can produce proof of falsified messages. Canetti et al. also extended the scheme through an interactive approach, to support receiver-deniability and combined them into a bi-deniable encryption scheme.

Numerous researchers have constructed translucent sets using a variety of methods based on this concept. Samplable encryption was implemented by Dürmuth and Freeman [4] to create a translucent set. A bi-translucent set built on a lattice was created by O'Neill et al. [5], in which they emphasize that the schemes are noninteractive and involve no third parties as they build bi-deniable public-key cryptosystems that allow both the sender and the recipient to communicate simultaneously.

Klonowski et al. [6] improved the Canetti et al. system to allow messages at any depth, by demonstrating that deniable encryption can be implemented in a different way,

so that it does not point to exploiting deniable encryption. Additionally, they demonstrated how even the initial purpose scheme could be expanded to permit any ''depth'' of deniability under certain conditions. In addition to translucent set approaches, other methods for creating deniable encryption algorithms have been proposed. For deniability, O'Neill et al. [5] used a voting strategy and a simulatable public-key system. An oblivious key generation function and an oblivious ciphertext function are offered by the simulatable public-key system. Another viable technique, proposed by Gasti et al. [7], has two pairs of public-private key pairings that the system claims to have created. The sender chooses which key is made available, based on the presence of a coercer. Ibrahim [8] used a scheme that relies on the quadratic residuosity of a two-prime modulus to provide deniability. A decryption method based on composite order groups was proposed by Chi and Lei [9]. In this approach, the real and predetermined counterfeit data are concealed in various subgroups of a composite order group.

A different notion of deniability was recently proposed by Canetti et. al. [10]: they presented a method that enables both the sender and receiver to make different claims. An external coercer cannot tell who is lying, using this strategy. The principles underlying encryption and authentication are sometimes the same, but deniable authentication is entirely distinct from deniable encryption. Deniable authentication is a technology that enables the sender and the receiver to authenticate one another and the ability to persuade a third party. The central concept of this technology is *Zero-Knowledge Proof*, which was developed by Dwork and Naor [11]. Meanwhile, numerous presented deniable authentication systems have been presented. Naor [12] provided ring signatures, together with a deniable ring authentication. Zhu et al. [13] suggested two deniable authentication systems, based on the discrete logarithm problem and the factoring problem. Fiat and Naor [14] first brought up the idea of broadcast encryption, an encryption technology that allows a portion of a universe of users to receive a message clandestinely. Attribute-based encryption (ABE) is a type of broadcast encryption. The security of personal information is frequently protected with ABE.

For outsourced decryption, Li et al. [15] suggested an ABE technique with full verifiability that can simultaneously verify the accuracy of transformed ciphertext belonging to authorised and unauthorised users. The standard model demonstrates that the proposed ABE technique with verifiable outsourced decryption is selectively chosen plain text attack (CPA-secure). Li et al. [16] provided a formal definition and security model for continuous leakage-resistant hierarchical attribute-based encryption (HABE). For usage with cloud storage systems, Li et. al. [17] offer a ciphertext-policy attribute-based encryption (CP-ABE) scheme with effective user revocation. The concept of the user group is introduced to address user revocation effectively. The idea is to transfer heavy computation loads to cloud service providers, without disclosing sensitive information and secret keys. Additionally, the system can withstand collusion attacks when banned users work with active users. Li et. al. [17] in the same work, has demonstrated that the suggested technique is secure under the Diffie-Hellman assumption of divisible computation, and local devices can compute with relatively little expense. To protect user privacy, Reddy et. al. [18] have provided a concept for a new cloud storage encryption system that enables cloud storage providers to produce unsuspicious fake user secrets. The cloud storage provider makes sure that user privacy is still securely maintained, because coercers are unable to determine whether the secrets they have obtained are real or fake.

The danger of the adversary detecting the fake message as such, in a scheme that uses only a single algorithm for honest and dishonest encryptions, was first addressed in [4]. They gave two examples of schemes of public-key bit encryption schemes. Complementary to this, we present a symmetric scheme whose security is based on entropy-considerations, but shares the same features by using (i) a single algorithm for fake and real messages, and (ii) addressing the fake detection problem by a formalization of plausible deniability. This, at the same time, addresses another issue that [4] brought up as the lack of formal proofs towards plausible deniability. We present a first one in this direction, showing that security remains accomplishable if we drop the otherwise usual assumption, that the attacker will recognize the correct message from a list of possible plaintexts [3].

Closely related to deniable- is also non-committing encryption, which differs from deniable encryption in letting the generation of (random) fake ciphertexts and corresponding randomizer such that these are indistinguishable from a legitimate encryption of the (true) information. Deniable encryption differs from this in enabling the user to create the randomness (in our case a key) that is suitable to decrypt the desired fake message. Deniable encryption, thus, also leads to non-committing encryption, but not conversely (see [1] for a respective example).

In the area of symmetric encryption, ambiguous multi-symmetric cryptography has been proposed [19] with goals similar to ours. They provide a discussion of various attack scenarios, including known-plaintext, chosen-ciphertext and others, all boiling down to the argument that this leaves the adversary with a linear system of equations that is under-determined. In basing their construction on number-theoretic arguments, they nonetheless rely on computational intractability (of finding factorizations or primes, though not via a usual reductionist argument). The computational complexity of this prior construction is governed by Chinese remaindering. Our scheme improves over this in requiring only linear efforts (a number of field operations that is proportional to the number of inner plaintexts).

An interesting additional security consideration is resilience against leakage of information from *access patterns*. In a series of papers [20], [21], [22], the focus of

plausible deniability constructions was shifted to hiding access patterns, as well as leveraging properties of the physical storage devices, using oblivious RAM among other techniques. Our discussion is agnostic of such physical features and does not hide access patterns as such, since the adversary model used here is only concerned with attacks on a static snapshot of the ciphertext, but allows partial side-information to be available to the adversary (similar as what access patterns could leak).

Extensions towards also preventing attacks via access pattern analysis are imaginable by means of oblivious RAM (ORAM) or private information retrieval. These techniques are also successfully employed by [23] to protect hidden volumes (like in the former TrueCrypt project, for which a detection of hidden volumes was demonstrable [24]). The work of [25] extends the possible adversaries to indexers and intermediate nodes for content delivery, which is not in scope of much other work on plausible deniability. Their approach is to prevent customer profiling by making access patterns themselves deniable. A similar shift of the adversarial perspective towards simultaneous compromise of the sender and the receiver in a deniably encrypted data transmission is adopted by [10], who rely on indistinguishability obfuscation and one-way functions. This work provides strong computational security.

Our formalization of security is information-theoretic, and based on residual uncertainty and the (assumed) ability to generate fake and real messages from the same (random) source. Techniques like generative adversarial networks (GANs) are natural to mention, but a particularly interesting work to support our assumptions is [26]: this reference is concerned with measuring the privacy risk when releasing sensitive data for public analysis. That is, with an attacker having unknown background information, [26] demonstrate the construction of "sanitized" information with quantifiable probability of data from individuals to be uniquely identifiable.

The authors of [6] improve on the early constructions of plausible deniability by Canetti et. al. [2] and discuss interesting additional possibilities like covert channels based on deniable encryption, and show how to enhance even standard encryptions like ElGamal with deniability (although not plausible deniability).

Generally, schemes can be divided into *plan-ahead* and *ad hoc* [6], [7], where for plan-ahead schemes, the user has to choose the fake messages in advance, whereas for ad hoc schemes, the fake messages can be generated at the moment when the honest party is put under adversarial pressure. Our distinction of deniability from plausible deniability in terms of statistical distributions put our construction into both categories: it is plan-ahead if we wish to open a plausible fake plaintext, but ad hoc if we only need deniability in the sense of opening any plaintext different from the real message (for example, if the encryption is for a session key that is just a random number).

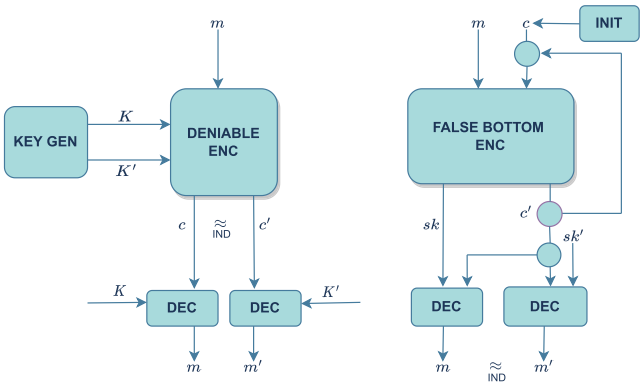

FIGURE 1. Conceptual comparison of Deniable encryption to False-Bottom encryption.

A. OUR CONTRIBUTION

This work offers the following novelties:

- We provide an information-theoretic security concept and a symmetric plausibly deniable cipher being secure without intractability assumptions (thus making the scheme post-quantum secure).
- We devise a scheme that is "editable" in the sense of letting us add, remove and change plaintexts within a ciphertext over time
- We provide a formal distinction and definition of deniability and plausible deniability, and generalize classical models (unicity distance) to prove these properties in light of background knowledge. This addresses an issue commonly defined with plausible deniability mechanisms in past implementations (like the now defunct TrueCrypt), which are considered as technical mechanisms without formal fundamental.

We show the construction of a symmetric and plausibly deniable encryption scheme from secret sharing called False-Bottom Encryption. The name "False-Bottom" alludes to well-known equipment of stage magicians, who have boxes that can be opened to appear empty or with something inside, hidden under a false bottom. Our scheme is designed for the same effect, hence the name. False-Bottom lets an encrypted data set (a ciphertext) be stored at some remote location, e.g., a cloud, with the secret key being with the user. If the user is coerced to reveal the encrypted data, the user can run the decryption algorithm with respectively modified versions of the secret key, such that the ciphertext c may decrypt into the real message m under a secret key sk, but also into a set of fake plaintexts $m_1, m_2, \ldots$ under respectively different keys $sk_1, sk_2, \ldots$. As shown in Fig. 1, if we use key sk to decrypt the ciphertext, it will give a message m, and the same ciphertext, when decrypted using $sk' \in \{sk_1, sk_2, \ldots\}$, will provide a different message $m' \in \{m_1, m_2, \ldots\}$.

We hereto distinguish the notion of "deniability" from "plausible deniability", where the latter accounts for possible background knowledge of the adversary, while the former is

TABLE 1. Comparison of our scheme to other encryption concepts.

Concept	Feature(s) / Purpose	vs	False-Bottom Encryption (Our Contribution)
Encryption	conceal a single plaintext		conceal multiple plaintexts
	encryption function's *outputs (ciphertexts)* are computationally indistinguishable → semantic security		encryption function's *outputs (ciphertexts)* will change their distribution over time → *no semantic security* in the cryptographic sense
	encryption function's *inputs (plaintext)* can have different distributions → plaintexts are *generally undeniable*		encryption function's *inputs (plaintexts)* are indistinguishable → *deniability of enforced decryptions*
	can be public-key or symmetric		symmetric (only)
(Fully) Homomorphic Encryption	allows plaintext updates by (arithmetic) addition and/or multiplication		allows additions, deletions and replacement of (any) inner plaintext
Deniable Encryption	multiple plaintexts decodable using different keys, but number of plaintexts deducible [2, 10]		same as deniable encryption, but number of (inner) plaintexts also uncertain for the attacker
Honey Encryption	equalizes plaintext distributions by preprocessing and lets a stream cipher follow [3], does not generally provide deniability, but decryption can emit plaintexts that look plausible		no preprocessing required, but needs a source of decoy messages with the same distribution as plaintexts. Can make use of plaintext distribution equalizing as in honey encryption

only about several possibilities of plaintext, but disregards their individual likelihood of being correct. For example, if the ciphertext c is from an enterprise in the car industry, then the expected plaintext m would relate to car manufacturing. If, a forced decryption of c (i.e. through coercion) results in another meaningful plaintext m' (which can be either $m_1, m_2, \ldots$), for example, cookie recipes, the adversary will see it as a strong sign that m' is not the real plaintext, and will hence keep up the force to decipher c into something else. We remark that normally, cryptographers are not concerned with the question of whether the attacker can recognize its results as correct; rather, a cryptanalysis is considered as successful, if the plaintext shows up (somewhere) in a list of deciphered information items, nontrivially narrowing down the list of all possible plaintexts [3].

We discuss how to formalize such background knowledge in terms of probability distributions and entropy, and adapt the (old) concept of unicity distance as a measure of how much background knowledge can be assumed until the deniability property deteriorates. Our use of unicity distance is motivated by the fact that our encryption, as constructed from information-theoretically secure mechanisms, is itself not hinging on computational intractability. Thus, its security will be formalized and proven in Shannon and Hellman's random cipher model and with entropies [27].

Cryptography, in many cases, defines security by *computational indistinguishability*, intuitively meaning that there is no polynomial-time algorithm that could classify a ciphertext to have been created from one or the other plaintext. This notion is equivalently called *semantic security* [28], and is the very basic security requirement imposed on any (probabilistic public-key) encryption. Our scheme, however, belongs to symmetric cryptography, to which semantic security generally does not apply as a security concept. We refrain from going into further details, except for highlighting how indistinguishability is understood as "equal probability distributions" is important in our setting. An encryption is "secure" if the *outputs* of an *encryption* function applied to two plaintexts $m \neq m'$ have computationally indistinguishable distributions, denoted by the $\overset{\approx}{\text{IND}}$ relation in Fig. 1 between c and c'. We seek the same property to hold for the *outputs* of *decryptions*, since our attacker forces us to decrypt. The easiest protection against this is imposing a fixed distribution on all plaintexts that are encrypted, denoted by $\overset{\approx}{\text{IND}}$ relation between m and m' in Fig. 1, so as to make correct and false decryptions indistinguishable.

This leads to a criterion for plausible deniability that asks how much background or context information would need to be known to the attacker, to single out the real plaintext among several ones extracted, all of which are meaningful, but not equally plausible. We will go into details about this in section VI-A4.

III. IDEA AND GENERAL OUTLINE

False-Bottom Encryption will hide a secret inside a set of existing data blocks. Let us assume that the secret that we wish to protect is a binary string. A "very large" such string will, if necessary, be split into smaller blocks, which we can (invertibly) map to elements from some finite field $\mathbb{F}$. Without loss of generality, we may thus assume our secret to be an element $m \in \mathbb{F}$. We will represent m as a weighted sum $m = r_1 \cdot \alpha_1 + r_2 \cdot \alpha_2 + \ldots + r_n \cdot \alpha_n$. Such a representation is trivial to find by choosing all random values $r_1, \ldots, r_n$ and $\alpha_1, \ldots, \alpha_{n-1}$, and solving the equation for the remaining value α_n. This representation is nothing else than a multivariate polynomial secret sharing, if we consider the r-values to be all constant. Applying this assumption, let us fix the set of r-values, and call it our "key-base" in the following. Representing m as a linear combination is, on its own, wasteful, since it expands a secret of the size of a field element into an n-fold (actually $2n$-fold) set of values; however, these values very well lend themselves to representing more (additional) secrets than m. For example, consider another secret m' to be represented likewise as m, but this time, we will re-use

r- and α-values from m's representation to write out $m' = r'_1 \cdot \alpha'_1 + r'_2 \cdot \alpha'_2 + \ldots + r_{n'} \cdot \alpha_{n'}$, with the convention that the *same values*, only in *different order and number* were used. That is, the r- and α-sets overlap $\{r'_1, r'_2, \ldots, r'_n\} \subseteq \{r_1, \ldots, r_n\}$ and $\{\alpha'_1, \ldots, \alpha'_{n-1}\} \subseteq \{\alpha_1, \ldots, \alpha_n\}$, leaving only $\alpha_{n'}$ to be computed afresh for m'. In making re-use of the variables for the representation of m', its inclusion expands the data by only a *single new element*, namely $\alpha_{n'}$. And this is already the core of the overall construction. The ciphertext is then defined as the set of α-values, which is $(\alpha_1, \ldots, \alpha_n, \alpha_{n'})$, in which all but the last values are relevant for both m and m', and only $\alpha_{n'}$ is exclusively computed for m'. Without knowledge of which r- and α values are to be used to recover m or m', both messages remain perfectly concealed. The "secret decryption key" is hence the information about (i) which and how many r-values are needed, plus (ii) which α-values are needed, and (iii) in which order they are to be multiplied and finally summed up to give the desired message. Further messages can be added in the same way, expanding the data by 1 element per new message (thus, the initial overhead becomes relatively less and less). Upon coercion, we can open up any fake message m' instead of the real message m, which delivers a form of symmetric deniable encryption. We make this intuition rigorous in the following.

IV. PRELIMINARIES AND NOTATIONS

In the following, we will let lower-case letters like t and k denote scalars, and bold printed letters like $\mathbf{c}$ and $\mathbf{r}$ be vectors. Typewriter font letters like $\mathtt{m}$ will represent strings. Uppercase letters in normal font denote sets and random variables. We will write $x \in_R M$ to mean a uniformly random draw of x from the finite set M. Likewise, the symbol $X \subset_R M$ denotes a choice of a subset of M, such that each $x \in X$ is sampled independently and uniformly distributed from M. If M is replaced by a vector, we analogously mean a choice of coordinates (treating the vector as an ordered set). The symbol $|M|$ will mean the size (cardinality) of a finite set, and we let $\dim(\mathbf{c})$ be the number of coordinates (the dimension) of a vector $\mathbf{c}$. Following the typical array-notation of (many) programming languages, we write $\mathbf{c}[i]$ to mean the i-th element of the vector $\mathbf{c}$.

The symbol $\mathbb{F}$ will denote an (arbitrary but fixed) finite field, with $\mathbb{F}^*$ being the subgroup of multiplicative units (being $\mathbb{F}^* = \mathbb{F} \setminus \{0\}$).

The symbol $X \sim F$ will hereafter mean a random variable X to have the distribution F. We write $H(X)$ to mean the Shannon-entropy of the random source X.

A. BASICS OF SECRET SHARING

Our work is an application of secret sharing. Secret sharing is the process of giving away a secret value m "in pieces", called *shares*, to several peers, called *players*, such that (i) no player can by itself infer the value of m based on the information in their possession, and (ii) a certain set of players is required to collaborate to reconstruct m by pooling their shares together. Shamir's scheme [29] uses polynomial interpolation, by choosing a random polynomial $p(x)$ of degree n with random coefficients, but such that $p(0) = m$. The shares that the players $1, 2, \ldots, n+1$ receive are the values $p(1), p(2), \ldots, p(n+1)$, and it is easy to see that the polynomial $p(x)$ is uniquely determined by $n+1$ or more values of it, but undetermined if $\leq n$ players put together their knowledge. In that case, the interpolation problem comes to an underdetermined system of equations $p(i) = m + r_1 i + r_2 i^2 + \ldots + r_n i^n$ for $i = 1, 2, \ldots, t < n$, leaving an attacker to determine $n+1$ unknowns from $\leq n$ linear equations. Blakley's [30] secret sharing is quite similar, yet gives a single linear equation in n variables to each player, which thereby possesses the description of an n-dimensional hyperplane. Like with the interpolation problem, if $t < n$ players pool their knowledge, they will jointly set up a system of t equations with $n > t$ unknowns, which is again underdetermined, and therefore will not allow the reconstruction of the secret m as the unique intersection of n hyperplanes in $\mathbb{R}^n$. Both schemes boil down to the idea of protecting the secret by leaving it to an underdetermined (linear) system of equations, unless there is enough information to make the system uniquely solvable. This is exactly the mechanism that we will utilize to protect the secrets in our deniable encryption scheme: the "degrees of freedom" that make the unknowns unambiguously determined, will be our secret decryption keys.

B. ENCRYPTION

An *encryption scheme* is a triple of algorithms (KeyGen, Enc, Dec), in which KeyGen is a probabilistic algorithm that takes one or more security parameters to initialize structures and algorithms accordingly, typically controlling bitlengths of blocks and keys. Additionally, it outputs a pair (sk, sk') as an encryption and decryption key. Since we will be concerned with symmetric schemes only, we may assume $sk = sk'$ hereafter, and speak about a "secret key" only (to distinguish our wording from public-key cryptography, where we would have public/private keys accordingly). The (possibly probabilistic) algorithm $\mathrm{Enc}(m, sk) = c$ maps a given message m under a secret key sk to a ciphertext c, with the (generally deterministic) algorithm $\mathrm{Dec}(c, sk)$ defined such that $\Pr(\mathrm{Dec}(\mathrm{Enc}(m, sk), sk) = m) = 1$ holds.

Definition 1 (False-Bottom Encryption): A *False-Bottom Encryption* is a tuple (Init, Enc, Dec)

- Init takes one or more security parameters and outputs all relevant system parameters, including initialization of algebraic structures and functions. In addition, it outputs an "empty" ciphertext $\mathbf{c}$.
- $\mathrm{Enc}(\mathbf{c}, m)$ is an algorithm that takes a "message m and current ciphertext $\mathbf{c}$" as input, and returns an updated ciphertext $\mathbf{c}'$ and a (new) decryption key sk_m to later decipher m from a $\mathbf{c}'$.
- $\mathrm{Dec}(\mathbf{c}, sk_m)$ is a decryption algorithm that takes the current ciphertext $\mathbf{c}$ and a secret key sk_m, and returns the message m (to which sk_m corresponds).

The main difference between a False-Bottom Encryption and a traditional setting of encryption is thus the generation of ciphertexts and keys at different times, as Fig. 1 illustrates. While a (standard) encryption scheme has an Init algorithm to produce the encryption keys uses Enc to generate ciphertexts under a given key, our scheme starts with an empty ciphertext instead of a key, and uses Enc to "insert" a message into the ciphertext, and also produces a respective decryption key along the process.

Besides the simple use of putting information into a ciphertext, we may also have practical use for manipulating the ciphertext; of course, contingent upon the knowledge of one (or more) secret keys. The usual practical method is using a (fully) homomorphic encryption; a simple example of which is [31], which also allows to decipher a string into several different plaintexts using different extraction secrets, but does not allow changing any content. Our construction is, however, essentially different from this, since the ability of putting new messages into a ciphertext $\mathbf{c}$ also lets us replace messages arbitrarily, provided that we have the respective decryption key.

Definition 2 (Editable False-Bottom Encryption): A False-Bottom Encryption (Init, Enc, Dec) is called *editable*, if it is an extended tuple (Init, Enc, Dec, Update, Del), with the following algorithms added:

- Update($\mathbf{c}, i, sk_1, sk_2, \ldots, sk_\ell, m'$): Assuming that we have ℓ plaintexts in $\mathbf{c}$ with corresponding secret keys $sk_1, \ldots, sk_\ell$, this algorithm takes the message index i and all secret keys $sk_1, \ldots, sk_\ell$ to replace the existing m_i by m'_i, returning a ciphertext $\mathbf{c}'$ such that (later) $m' \leftarrow \mathrm{Dec}(\mathbf{c}', sk_i)$. The purpose of using i as an index here is to distinguish among the various messages as if $i = 1$ represents m_1; likewise, $i = 2$ represents m_2 and so on.
- Del($\mathbf{c}, i, sk_1, \ldots, sk_\ell$): this function removes a plaintext m (as would be returned by $\mathrm{Dec}(\mathbf{c}, sk)$) from $\mathbf{c}$, given its decryption key sk, and returns an updated ciphertext $\mathbf{c}'$. The new ciphertext $\mathbf{c}'$ is such that $\mathrm{Dec}(\mathbf{c}', sk) \neq m$.

The following will be a construction of an editable False-Bottom Encryption as defined above.

V. THE CONSTRUCTION

We let $t \in \mathbb{N}$ be a parameter that determines the block size of our cipher, and let another parameter n be the number of blocks to initialize the ciphertext. A third parameter $k \in \mathbb{N}$ will determine the common size of the secret key sk for the message m. We will use an index to distinguish message symbols and relate them to their respective encryption and decryption keys. The values t, k and n are chosen independently of one another, but of individually reasonable sizes for efficiency, i.e., the value t will be large, while the values of n and k may be small initially.

A. SYSTEM INITIALIZATION $(\mathbf{c}, \mathbf{r}) \leftarrow \mathrm{Init}(t, n, k)$

Given the security parameters t and n, we construct a finite field $\mathbb{F} = \mathbb{Z}_p[X]/(f(X))$, where p is a prime, and f is an irreducible polynomial of degree $d \in \mathbb{N}$, chosen such that $\lceil \log_2(p) \rceil \cdot d \geq t$. In this way, an element $x \in \mathbb{Z}_p$ can act as a single block, and a whole string $\mathtt{m} \in \{0,1\}^n$ can be split up into blocks $\mathbf{m} = (m_1, \ldots, m_d) \in \mathbb{Z}_p^d$ of up to $\log_2(p)$ bits, each of which fits into the field $\mathbb{Z}_p$. Hence, the whole message $\mathbf{m}$ can be encoded into an element from $\mathbb{F}$. We hereafter use different font faces to distinguish when m is a string (denoted by typewriter letters $\mathtt{m}$), or a vector (bold-printed $\mathbf{m}$) or a field element (normal font m). Note that the number of bits to encode an element $m \in \mathbb{F}$ is at most t, so that we can associate any string $\mathtt{m} \in \{0,1\}^t$ invertibly with a field element $m \in \mathbb{F}$. The *key base* (or root-key) is chosen as a set of nonzero values $\{r_1, \ldots, r_k\} \subset_R \mathbb{F}^*$ that we store in an arbitrary but fixed order in a vector $\mathbf{r} = (r_1, \ldots, r_k)$. This ordered set will remain constant over time, and will serve as a basis to draw elements when adding new plaintexts to a (growing) ciphertext set.

Remark 1: For messages of practical size in the range of kilobytes, megabytes or larger, we can hardly expect Galois fields of feasible size to take up a whole message, so we will assume that large messages are split into respectively smaller blocks (according to electronic codebook mode or more advanced block cipher modes), each of which becomes another message to be added.

Finally, we initialize an "empty" ciphertext as a sequence of k random values, stored in fixed order in a vector $\mathbf{c} = (\alpha_1, \ldots, \alpha_k) \in_R \mathbb{F}^k$.

To ease notation, we hereafter assume that all system parameters are available to all functions, and subject to the usual security requirements, which are authenticity and non-malleability, not including confidentiality in most cases, since we assume a cryptanalyst to know all system parameters, algorithm details, but not the secrets. The latter are hence handled as input and output parameters of the respective algorithms.

B. ADDING MESSAGES TO THE (EXISTING) CIPHERTEXT $(\mathbf{c}', sk_1, \ldots, sk_\ell, sk_{NEW}) \leftarrow \mathrm{Enc}(\mathbf{c}, m, sk_1, \ldots, sk_\ell)$

Since we are anticipating to add several messages $m_1, m_2, m_3, \ldots$ to the ciphertext $\mathbf{c} = (\alpha_1, \ldots, \alpha_{\dim(\mathbf{c})})$ over time, we hereafter let each message carry an index i (like an identity), which uniquely associates it with the set of decryption keys to be generated here. Therefore, let us assume that we want to add the message $m_i \in \{0,1\}^t$ to the (initially empty) ciphertext $\mathbf{c}$ by calling $\mathrm{Enc}(\mathbf{c}, m_i)$ with a yet empty list of secret keys. This triggers the following steps: We pick a number $n_i \in_R \{2, \ldots, k\}$ and sample $n_i - 1$ indices $(j_{i,1}, \ldots, j_{i,n_i-1}) \in_R \{1, \ldots, \dim(\mathbf{c})\}^{n_i-1}$, which set the values $\alpha_{i,j} \leftarrow \mathbf{c}[j]$ for all $j \in \{j_{i,1}, \ldots, j_{i,n_i-1}\}$. Then, we draw another independent set of n_i values at random indices $\rho_{i,1}, \ldots, \rho_{i,n_i}$ within the key space, and set up a

linear equation to represent m_i with $r_{i,j} \leftarrow \mathbf{r}[\rho_{i,j}]$ for $j = 1, 2, \ldots, n_i$ as

$$m_i = \alpha_{i,1} \cdot r_{i,1} + \alpha_{i,2} \cdot r_{i,2} + \ldots + \alpha_{i,n_i-1} \cdot r_{i,n_i-1} + \alpha \cdot r_{i,n_i}, \tag{1}$$

in which α is the only unknown and (easily) computable to make (1) hold. The updated ciphertext is then returned as $\mathbf{c}' \leftarrow \mathbf{c} \| \alpha$, where $\|$ means that we straightforwardly append the just computed value to the list of values in $\mathbf{c}$.

The secret key returned to recover m_i from $\mathbf{c}'$ is simply the information on which indices in the vector (list) $\mathbf{c}'$ and which elements of the key base $\mathbf{r}$ were used to represent m_i. That is, we return the secret key as the list of pairs $sk_{\text{new}} := ((j_{i,1}, \rho_{i,1}), \ldots, (j_{i,n_i-1}, \rho_{i,n_i-1}), (\dim(\mathbf{c})+1, \rho_{i,n_i}))$. The last index in the ciphertext is herein always the length of the ciphertext $+1$, since we just append the value to $\mathbf{c}$. Note, however, that after adding further messages to $\mathbf{c}$, before it comes to a possibly forced decryption, this currently last index will later no longer be the last element.

Since we explicitly encode indices and not the actual values, the size of the key is at most logarithmic in the size of the current ciphertext, and grows at a practically feasible rate: we have no more than k terms involved in (1), each ρ-value taking up $O(1)$ bits (since the size of $\mathbf{r}$ is fixed), and each entry in $\mathbf{c}$ occupying $O(1)$ bits, since it is an element of the fixed finite field $\mathbb{F}$. The overall key size thus comes to $O(k \cdot \log \dim \mathbf{c}) = O(\log \dim(\mathbf{c}))$ bits, since the index range will become larger as new plaintexts are added.

C. DECRYPTION OF THE i-TH MESSAGE $m_i \leftarrow \mathsf{Dec}(\mathbf{c}, sk_i)$

Let $\mathbf{c}$ be the "current" ciphertext (e.g., from a previous call to Enc; see above). To decrypt the i-th message, we extract the respective indices from sk_i and recover m_i via equation (1). The correctness is trivial, but the security deserves a closer look. Before that, however, let us additionally discuss possibilities to modify and delete messages from $\mathbf{c}$.

D. CHANGING PLAINTEXTS INSIDE $\mathbf{c}$ $\mathbf{c}' \leftarrow \mathsf{Update}(\mathbf{c}, m, sk_1, \ldots, sk_\ell, m')$

Since we are frequently re-using parts of the ciphertext to represent new plaintexts, any modification of the ciphertext towards updating some specific m_i will inevitably affect (possibly many) other plaintexts as well. Hence, the process of changing a previously encrypted message is tied to the way in which we select the elements of $\mathbf{c}$ to represent it. To formalize this, it is convenient to reshape the list of pairs that make up a secret key sk_i for the message m_i as a matrix with two rows, where the upper row, denoted as $\mathbf{sk}_i^{(1)}$ contains indices pointing into the vector $\mathbf{c}$, and the lower row, denoted as $\mathbf{sk}_i^{(2)}$ giving the indices related to the key-base $\mathbf{r}$, i.e., write

$$sk_i = \begin{pmatrix} \mathbf{sk}_i^{(1)} \\ \mathbf{sk}_i^{(2)} \end{pmatrix} = \begin{pmatrix} j_{i,1} & j_{i,2} & \cdots & j_{i,n_i} \\ \rho_{i,1} & \rho_{i,2} & \cdots & \rho_{i,n_i} \end{pmatrix}$$

We include the indication of i here to distinguish different keys for distinct messages.

If sk_i is such that it contains an element from $\mathbf{c}$ that is used for m_i, but for no other message $m_j \neq m_i$, then we can directly alter m_i by changing the respective part of the ciphertext accordingly. A sufficient condition for editability thus comes to the requirement that:

$$\text{for all } i \text{ we need to have } \mathbf{sk}_i^{(1)} \setminus \bigcup_{j \neq i} \mathbf{sk}_j^{(1)} \neq \emptyset, \tag{2}$$

possibly allowing for more than one element in $\mathbf{c}$ to be used for m_i only. The next lemma answers the question of whether (2) is accomplishable within the scheme set up so far (i.e., it is an invariant if Enc is respectively implemented):

Lemma 1: For putting a new plaintext m into $\mathbf{c}$, it is always possible to select $n \in_R \{2, \ldots, k\}$ elements from $\mathbf{c}$ such that condition (2) holds for all previous messages, and the new message.

Proof: The proof is by induction over the number ℓ of messages in $\mathbf{c}$. At $\ell = 1$, i.e., for the first message m_1, matters are trivial, since no other message exists that could share parts with m_1. Furthermore, note that the empty ciphertext initially has size k, and grows to size $k + 1$, by adding a new element to represent m_1. Our first claim (hence already established in the induction start) is that $\dim(\mathbf{c}) \geq 1 + k$. We hereafter need to prove two things, namely that (2) holds, and – to accomplish this without changes to the scheme as described – the selection of a random number of at least 2 and no more than k elements is doable without violating condition (2).

For the inductions step, let $m_{\ell+1}$ be the message to be added, and assume that we have (i) already added ℓ messages, and (ii) the size of the ciphertext is $\dim(\mathbf{c}) \geq \ell + k$. It is easy to directly see the latter condition to hold, since upon each message, the length of $\mathbf{c}$ grows by 1 and starts from size k, so that the latter condition already holds (by induction).

From (2) as our induction hypothesis, we can, for each element $m_1, m_2, \ldots, m_\ell$ select a (uniquely associated) element $c_1, \ldots, c_\ell \in \mathbf{c}$. We collect these elements in a set F and consider them as "forbidden" in the selection of new elements to represent $m_{\ell+1}$. This assures that (2) continues to hold. Since Enc is stateful, it is a simple matter of bookkeeping (done by inputting and outputting all the secret keys to the procedure when we add new plaintexts) to remember these "forbidden" elements, so that we can let the inner random choice of Enc be on the set $\mathbf{c} \setminus F$, i.e., it can choose from all elements in $\mathbf{c}$, except those that are forbidden (the set F). The size of the pool to choose elements from is thus $\dim(\mathbf{c}) \geq \ell + k - |F| = \ell + k - \ell = k$, thus leaving the desired choice of up to k elements possible. This completes the induction step. □

Changing a message m_i into m_i' is hence easy: by condition (2), we can find an element $\alpha_i^* \in \mathbf{c}$ that only appears in the representation of m_i via (1). Setting up the equation to yield m_i' and solving for the respective new α to replace the existing part in $\mathbf{c}$ accomplishes the update $m_i \leftarrow m_i'$.

E. DELETING MESSAGES FROM $\mathbf{c}' \leftarrow \mathsf{Del}(\mathbf{c}, i, sk_1, \ldots, sk_\ell)$

There are several options to delete the i-th plaintext m_i from $\mathbf{c}$, and we discuss them with deniability in mind:

- simply delete the decryption key for m_i. This is the discouraged method, since it leaves m_i to exist further in $\mathbf{c}$, and will still appear in a list of brute-force decrypted results.
- identification of some $\alpha \in \mathbf{c}$ that only m_i uses in its representation, and deleting it from $\mathbf{c}$. This requires according updates to all decryption keys (as the indices past $\alpha \in \mathbf{c}$ accordingly shift), and makes $\mathbf{c}$ smaller. While this irrecoverably removes m_i from $\mathbf{c}$, the shrinking of $\mathbf{c}$ is at the same time an indication that the deletion was done. Hence, if the adversary observes that $\mathbf{c}$ becomes shorter, we cannot deny the deletion as such, although we still can deny any plaintext that has been recovered.
- overwriting m_i with random data by "editing" it like described above. This also permanently removes m_i from $\mathbf{c}$, and makes the delete-operation indistinguishable from the edit-operation. Furthermore, it leaves all keys sk_j for $j \neq i$ intact, as the price of letting $\mathbf{c}$ never becomes smaller.

The deletion is thus merely the following procedure: choose a random element $m^* \in_R \mathbb{F}$ and overwrite the i-th message with it by calling $\mathbf{c}' \leftarrow \mathsf{Del}(\mathbf{c}, i, sk_1, \ldots, sk_\ell) = \mathsf{Update}(\mathbf{c}, i, sk_1, \ldots, sk_\ell, m^*)$. Additionally, remove sk_i from the list of secret keys that the data owner maintains (securely).

F. A NUMERIC EXAMPLE

The programming is done on the Python 3 Jupyter Notebook.[1] To simplify matters (cf. Remark 1), we consider only one plaintext block as a message in the following.

We have initialized the ciphertext $\mathbf{c}$ of size (dimension) $\dim(\mathbf{c}) = k = 5$ and the values of ciphertext $\mathbf{c} = (\alpha_1, \ldots, \alpha_k) = (6255, 2736, 6974, 7804, 2675)$. The key space of size $k = 5$ is initialized with values $\mathbf{r} = (454, 409, 1814, 1167, 2906)$. Let the message $m_1 = 100$, picking $n_1 = 3$ and assigning the indices $(\rho_{1,1}, \rho_{1,2}, \ldots, \rho_{1,n_1=3}) = (1, 2, 4)$ and their respective values are $(r_{1,1}, r_{1,2}, \ldots, r_{1,n_1}) = (409, 1814, 2906)$. Choosing $(n_1 - 1)$ random values $(\alpha_{1,1}, \ldots, \alpha_{1,j}) = (7804, 6974)$, we can compute the new α to be

$$\alpha = r_{1,n_1}^{-1} \cdot \left(m_1 - \sum_{j=1}^{n_1-1} \alpha_{1,j} r_{1,j} \right) = 7459$$

Therefore the α-values to represent m_1 are $(\alpha_{1,1}, \ldots, \alpha_{1,n_1}) = (7804, 6974, 7459)$. Now we will update the ciphertext and output $\mathbf{c}' = (6255, 2736, 6974, 7804, 2675, 7459)$. For the decryption of m_1, the ciphertext indices $(j_{1,1}, \ldots, j_{1,n_1}) = (3, 2, 5)$ and the key indices $(\rho_{1,1}, \rho_{1,2}, \ldots, \rho_{1,n_1}) = (1, 2, 4)$ forms the decryption key $sk_1 = ((3, 1), (2, 2), (5, 4))$. Putting the values of sk_1 in equation (1) we recover the message $m_1 = 100$.

Now, let us add another message to the ciphertext being $m_2 = 200$. We pick $n_2 = 3$ and assign the indices of $(\rho_{2,1}, \rho_{2,2}, \rho_{2,3}) = (0, 2, 3)$ and respective keyspace values are $(r_{2,1}, r_{2,2}, r_{2,3}) = (454, 1814, 1167)$. Choosing $n_2 - 1$ random values $(2675, 7804)$ from $\mathbf{c} = (\alpha_1, \ldots, \alpha_N) = (6255, 2736, 6974, 7804, 2675, 7459)$, putting these values in

$$\alpha = r_{2,3}^{-1} \cdot \left(m_2 - \sum_{j=1}^{n_2-1} \alpha_{2,j} r_{2,j} \right)$$

and computing the value of $\alpha = \alpha_{2,N+1} = 5587$. Therefore, the α-values to represent m_2 are $(\alpha_{2,1}, \ldots, \alpha_{2,n_2}) = (2675, 7804, 5587)$. Now we will update the ciphertext again to output $\mathbf{c}' = (6255, 2736, 6974, 7804, 2675, 7459, 5587)$. For the decryption of m_2, the ciphertext indices $(j_{2,1}, \ldots, j_{2,n_2}) = (4, 3, 6)$ and the key indices $(\rho_{2,1}, \rho_{2,2}, \rho_{2,3}) = (0, 2, 3)$ forms the decryption key $sk_2 = ((4, 0), (3, 2), (6, 3))$. Putting the values of sk_2 in equation (1) we get our desired message $m_2 = 200$.

VI. EFFICIENCY AND SECURITY ANALYSIS

Hereafter, we use asymptotic Landau symbols only for convenience, to avoid having to deal with redundancy information used, or parsing ciphertexts or keys. The constants inside the O notation will be due to separator symbols and encodings, to enable an extraction of the elements of vectors or lists, and as such, can be expected to be small.

By construction, the empty ciphertext has length k, and grows by 1 element per new plaintext being added. For ℓ plaintexts in total, we thus have $\dim(\mathbf{c}) \in O(k + \ell) = O(\ell)$. Editing and deleting messages from $\mathbf{c}$ does not change this length (it only possibly reduces it), leaving the bound unchanged.

For the *relative overhead* of the ciphertext size versus the plaintext size, we assume a block size of n bits per message, and the ciphertext as a vector of length $O(\ell)$ over a field $\mathbb{F}$ has thus a length of $O(n \cdot \ell)$ bits, whereas a set of ℓ plaintexts comes to $\ell \cdot n$ bits in total. Letting η be the constant in the $O(n \cdot \ell)$, the relative overhead is thus $\frac{\text{size of ciphertext}}{\text{total size of all plaintexts}} \leq \frac{\eta \cdot \ell \cdot n}{\ell \cdot n} = \eta \in O(1)$, so the scheme is asymptotically efficient (indeed, the constant overhead is bounded by the size of the empty ciphertext to be k elements from $\mathbb{F}$).

The size of secret key material is, for ℓ messages in total, upper bounded by $O(\ell \cdot \log \ell)$ for the indices pointing into $\mathbf{c}$, and $O(\ell \log k)$ for the indices pointing into $\mathbf{r}$. Since k is a constant, we have a space complexity of $O(\ell \cdot \log \ell)$ to encode all secret keys $sk_1, \ldots, sk_\ell$.

A. SECURITY ANALYSIS

We distinguish different scenarios under which we analyze security. We start with an attacker knowing only the cipher-

[1] We remark that in the following, index *values* will be zero-based (due to our use of the Python language), as opposed to the 1-based indexing used in the textual mathematical description.

text, but being able to "force" the legitimate user to use a decryption key to reveal what is inside the ciphertext **c**. Next, we analyze an offline attack, assuming the adversary has infinite computational power, and in the third case, we add some background knowledge to this (unlimited) attacker.

We describe the respective security models and definitions in each case separately, introducing the respective attacker model and security definition along with the scenario.

1) FORCED DECRYPTIONS

In this situation, the attacker cannot access the secret key, but it can make the user provide a key to decipher the message. Let $m_1, \ldots, m_\ell$ be all of Alice's messages, among which $m^* \in \{m_1, \ldots, m_\ell\}$ is the real secret, and all $m_i \neq m^*$ are the fake secrets to be revealed if necessary for deception. We will model the honest user as an algorithm with secret internal variables (in the jargon of object-oriented programming, one may think of a class with private members) that are the secret keys. Force brought upon the user by the attacker is modeled by letting the attacker be an algorithm with oracle access to U.

Definition 3 (Adversary and Security Model for Forced Decryptions): Let **c** be a ciphertext encapsulating a real message m^* and $\ell \geq 1$ fake messages $m_1, \ldots, m_\ell$. Let $sk^*, sk_1, \ldots, sk_\ell$ be the decryption keys that the honest user knows but which are unknown to the attacker. Let the honest user be an algorithm defined as follows:

procedure $U(\mathbf{c})$

1: $sk \in_R \{sk_1, \ldots, sk_\ell\}$ ▷ $sk \neq sk^*$
2: **return** Dec(**c**, sk)

The attacker putting *force* on the user is an algorithm $\mathcal{A}_F^U$ with oracle access to (the user) U, and we define the residual uncertainty about m^* as our security measure, i.e., we define

$$\mathcal{A}_F^U(\mathbf{c}) := H(m^* \mid m \leftarrow U(\mathbf{c})), \tag{3}$$

where $H(\cdot \mid \cdot)$ is the conditional Shannon-entropy.

We call an encryption *secure against forced decryptions*, if $\mathcal{A}_F^U(\mathbf{c}) > 0$ for all **c**.

Theorem 1: The encryption scheme from Section V is secure in the sense of Definition 3.

Proof: Assume that the attacker forces Alice to reveal several messages until the theoretical maximum possible number. How many messages can the attacker expect to be hidden inside **c**, when **c** is arbitrary? From the construction, it knows that each new message expands the ciphertext vector by one new element, and the initial length was somewhere in the range $2, 3, \ldots, k$. Therefore, the total number of messages hidden in **c** is *deniable* by Alice, claiming to any number $2 \leq n_1 < \ell \leq k$ of messages to be hidden inside **c**.

With the algorithm U secretly choosing sk from the set of at $\ell \geq 1$ decoy keys, the attacker receives $m \leftarrow U(\mathbf{c})$, and since m is stochastically independent of m^*, we have $H(m^* \mid m) = H(m^*) > 0$, since m^* is unknown to the attacker. □

This attack assumes force to be put only once on the user, but not multiple times. Equation (3) could be generalized to allow a number of > 1 messages retrieved from U. The argument from the proof of Theorem 1 remains intact up to the point where all $\ell + 1$ messages (i.e., all fake messages and the real message m^*) were obtained from U and considering $H(m^* \mid m_1 \leftarrow U(\mathbf{c}), m_2 \leftarrow U(\mathbf{c}), \ldots)$ (remember that U is a randomized algorithm), in which case eventually $H(m^* \mid m_1, m_2, \ldots, m^*) = 0$, provided that the adversary reliably recognizes m^* as the real message among $\{m_1, m_2, \ldots, m^*\}$.

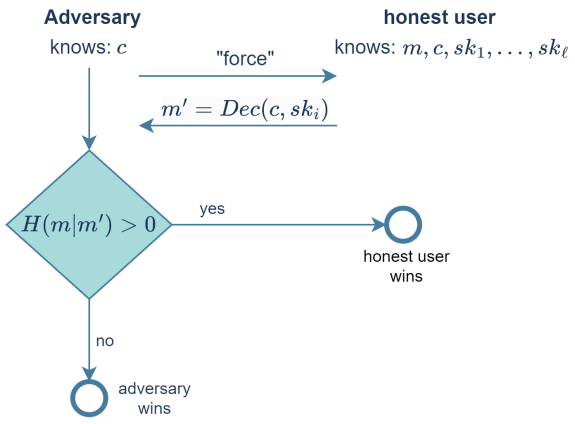

FIGURE 2. Forced decryptions.

The ultimate success of this attack will therefore depend on whether or not the attacker can be convinced that any of the fake messages that Alice provides through her keys is "plausible" as shown in Fig. 2. This will depend on the background knowledge of what the adversary expects to see as a plaintext, and will be formalized in Section VI-A4. Generally, security against brute forcing thus holds only for a single action of force, or in case of fake messages being (practically) indistinguishable from the real message. Otherwise, the security in the sense of Def. 3 deteriorates.

2) OFFLINE BRUTE-FORCE ATTACKS

Now, assume that the attacker has access to the ciphertext and attempts a brute-force trial decryption of some message m_i, but in absence of Alice, so she cannot be forced to participate. Similar as before, we will formalize this attack by the residual uncertainty of the attacker, conditional on all information obtained by running through the entire key space.

Our goal is to show that the number of guesses is larger than the brute-force complexity to guess the decryption key, as given by

Proposition 1: The brute-force complexity of decrypting all messages from **c** is given by

$$k \cdot p^d + \sum_{n_i=2}^{k} \binom{k}{n_i} \cdot \frac{\dim(\mathbf{c})!}{(\dim(\mathbf{c}) - n_i)!} \in \Theta(k^{\dim(\mathbf{c})}) \tag{4}$$

Proof: Given the vector **c** of length $N = \dim(\mathbf{c})$, it knows that m_i is composed from a random selection of $n_i \in \{2, 3, \ldots, k\}$ many elements from **c**, weighted with a selection of the same number of random values $r_1, \ldots, r_k \in$

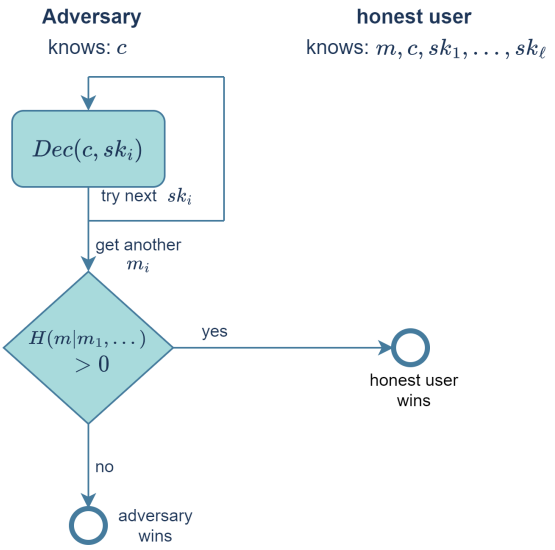

FIGURE 3. Offline Brute-Force attacks.

$\mathbb{F}$. The number of choices from $\mathbf{c}$ is a selection of n_i out of N, without replacement and with order, which is $\frac{N!}{(N-n_i)!}$. Additionally, we have $n_i = 2, 3, \ldots, k$ as possible choices, and a fixed lot of $k \cdot |\mathbb{F}| = k \cdot p^d$ additional independent bits to guess for the coefficients in $\mathbf{r}$. Since $\mathbf{r}$ is fixed, let us assume that the attacker, for efficiency, first guesses the entire $\mathbf{r}$, and is then only left with a choice about the indices from $\mathbf{r}$ for the message m_i, which is $\binom{k}{n_i}$ many choices. The complexity of guessing $\mathbf{r}$, plus the guessing of n_i and the subset of $\mathbf{c}$ gives the number as claimed above, where the summation comes from the attacker's uncertainty about how many terms (at least 2 but up to k) are required to reconstruct the message. It is a matter of straightforward calculations to see that the total count is within $\Theta(k^{\dim(\mathbf{c})})$, in which only the dimension, i.e., length, $\dim(\mathbf{c})$ is variable over time, and k (and all other variables) are constants. □

Proposition 1 shows that the brute-force complexity of exhaustive key search is growing *exponentially* with the size of the ciphertext and thus beyond polynomial time-bounds if we assume a polynomially bounded adversary. Formalizing security in terms of residual entropies and without assumed bounds on the computational power, we can introduce the following security model and definition:

Definition 4 (Security against Offline Brute Forcing): Let $\mathbf{c}$ be a ciphertext of dimension $\dim(\mathbf{c})$, encapsulating a real message $m^* \sim F$ and ℓ decoy messages $m_1, \ldots, m_\ell \sim F$, all sampled from the same (plaintext) source distribution F. Let K be the key space for the encryption. Let the adversary be a computationally unbounded algorithm and define

$$\mathcal{A}_{BF}(\mathbf{c}) := H(m^* \mid \{m_{sk} \leftarrow \mathrm{Dec}(\mathbf{c}, sk)\}_{sk \in K}).$$

We call an encryption *secure against offline brute-force*, if $\mathcal{A}_{OBF}(\mathbf{c}) > 0$, if despite all information extractable by trial encryptions, there remains residual uncertainty about the real message m^*.

Against this definition, we can prove the following:

Theorem 2: The encryption scheme from Section V is secure in the sense of Definition 4.

Proof: Once the attacker has made the number (4) of guesses and corresponding trial decryptions, it has a list of $m_1, \ldots, m_\ell$ messages that Alice has encoded into $\mathbf{c}$. In a traditional setting, the attacker would be considered successful if Alice's real secret m^* is in this list, which it definitely is. However, since we are after deniability in this work, we need to consider the adversary's residual uncertainty about which of the $m_1, \ldots, m_\ell$, among which there is m^*, is Alice's real secret. This is an additional choice of $H(m^* \mid \{m_{sk} \leftarrow \mathrm{Dec}(\mathbf{c}, sk)\}_{sk \in K}) = \mathcal{A}_{OBF}(\mathbf{c}) = \lceil \log_2 \ell \rceil$ bits to guess, since all m_{sk} (among which m^* is) have identical and hence indistinguishable distributions (irrespectively of computational power). This residual uncertainty establishes security in the sense of Definition 4, since it adds to the attacker's workload, and hence exceeding the brute-force complexity accordingly as shown in Fig. 3. □

Similarly as before, the security is computational in the sense that there is a finite number of keys to try, if the attacker can recognize the correct real message m^* among whatever it obtains to this end. Against unbounded attackers, we must additionally assume indistinguishable fake and real messages, which we formalize and study in Section VI-A4. It is possible to assume (only) computational indistinguishability here, in which case the security again falls back to computational.

This assurance remains, however, crude in the sense that the attacker may have background knowledge upon which it can rule out some (up to $\ell - 1$) messages as implausible, so as to single out m^* without much additional effort. We dedicate section VI-A4 to a closer look at what the availability of background information can change here, looking first at how information about the ciphertext itself can help.

3) ATTACKER WITH KNOWLEDGE ABOUT THE HISTORY OF $\mathbf{c}$

Suppose that the attacker is able to track changes to the ciphertext $\mathbf{c}$ over time, e.g., if the file system to store $\mathbf{c}$ has a journaling function activated, or if the adversary acquires two backup versions of the cloud storage. Forensic investigations of the data may bring up such knowledge. Tracking the change, the attacker may thus know about some particular α-values in $\mathbf{c}$ that have definitely been used in a recent file that was stored inside $\mathbf{c}$. The question studied now is how much this information may help, in the sense of reducing the attacker's uncertainty about the hidden plaintext.

The change track of $\mathbf{c}$ provides the adversary with partial information on which α-values are relevant for the i-th message that went into $\mathbf{c}$, where the adversary can easily track how many messages were added. Furthermore, it can recognize manipulations of existing messages in $\mathbf{c}$, although (if the deletion is done by overwriting), the attacker cannot distinguish whether Alice has edited or deleted a message. Overall, the information gained in this way reduces the brute-force

complexity to break the ciphertext down to

$$k \cdot p^d + \sum_{n_i=2}^{k} \binom{k}{n_i} \cdot \frac{(k+(\ell-\nu))!}{(k+(\ell-\nu)-n_i)!} \tag{5}$$

where $(k+\ell)$ is the length of the ciphertext $\mathbf{c}$, and ν is the number of blocks that the attacker knows to have changed and hence have been used to put new messages inside, therefore the dimension of the ciphertext is reduced to $(k+(\ell-\nu))$. However, we have $n_i = 2, 3, \ldots, k$ as possible choices and $k \cdot p^d$ independent bits to guess for the coefficients in $\mathbf{r}$. This makes the system resistant to brute force attacks because it is difficult for the attacker to guess, even having the knowledge about the history of $\mathbf{c}$.

The background knowledge thus reduces the size $|K|$ of the key space to search, but the number (5) remains finite and positive, so Theorem 2 applies likewise.

4) ATTACKER WITH BACKGROUND KNOWLEDGE (INFINITE COMPUTATIONAL POWER)

Let us assume that the attacker has infinite computational power, and can thereby disclose all m_i that went into $\mathbf{c}$. Whether or not it will be possible to determine which of the m_i's has been the real message now depends on the attacker's background information about the unknown message m^*.

Definition 5 (Deniability, Plausible Deniability (generic definition)): We say that an encryption scheme is:

- *deniable*, if for every given ciphertext c, there are keys sk_1, sk_2 such that $c = \text{Enc}(m_1, sk_1) = \text{Enc}(m_2, sk_2)$ for meaningful messages $m_1 \neq m_2$ (the messages may herein come from different random sources).
- *plausibly deniable*, if it is deniable, and the messages m_1, m_2 are sampled from the same probability distribution F, i.e., m_1, m_2 are samples from a common random variable $M \sim F$, so that meaningful and meaningless messages are indistinguishable.

The above definition is intentionally "generic" in the sense of not being explicit on an encryption scheme to be specified as a triple or larger tuple of algorithms, so it can be concretized for different forms of encryption (conventional, but also extended, as we study in this work). Also, the notion of "indistinguishability" may be instantiated more concretely to be computational, statistical or perfect, where the latter is here understood as "having the same distribution". We will, in the following, consider the latter variant, and not resort to computational or statistical (approximate equality) indistinguishability. To formalize and analyze the degree to which our scheme can satisfy Definition 5, we adapt the concept of the *unicity distance* [32]. It will be instructive for the analysis to review the ideas of unicity distance and the random cipher model for our purposes accordingly. To this end, let us, for the moment, introduce and consider the random variables K, M and C, from which the secret keys, secret plaintexts and resulting ciphertexts are sampled (resp. computed). Unicity distance, in the way Claude Shannon [27] introduced it, measures the amount of ciphertext that is required to pin down the decryption key uniquely. Intuitively, it is the minimum size of the information C such that the conditional entropy $H(K|C)$ about the decryption key $sk \sim K$ vanishes. We have an analogous problem here, since we may ask for the minimum amount of information that the adversary needs to know about the plaintext, so as to rule out other meaningful plaintexts for implausibility by inconsistency with the attacker's background information.

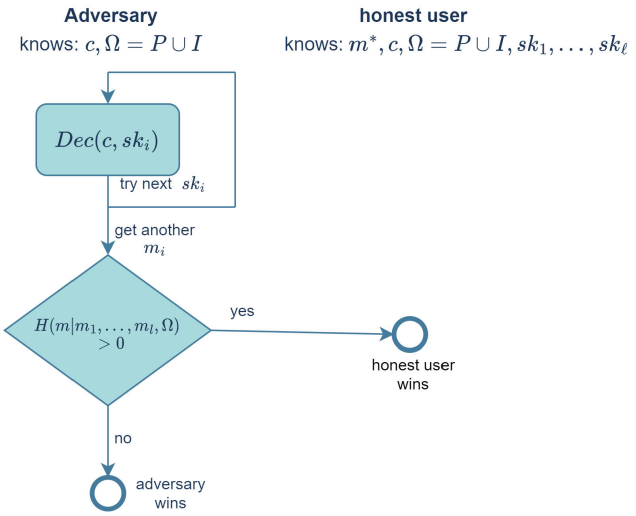

FIGURE 4. Attacker with background knowledge.

To approximate the unicity distance, we may let the attacker do trial decryptions (hence the name "random cipher model"), and assume that the decryption function $\text{Dec}(\cdot, k)$, using the trial key k, is such that the value $\text{Dec}(c, k)$ for the given ciphertext c is a random variable that is uniformly distributed over the set of all (meaningful and meaningless) plaintexts. The unicity distance is the smallest length of c (in number of blocks) such that a computationally unbounded attacker could recover the decryption key. For block ciphers under known ciphertext attacks by random trial decryptions, it is known [Fact 7.71] [27] that the expected unicity distance is $H(K)/D$, where $H(K)$ is the key entropy, and D is the language's redundancy. The argument to prove this nicely lends itself to our purposes as well.

Let c be a ciphertext, and let a computationally unbounded attacker have recovered possible meaningful plaintexts $m_1, m_2, \ldots$ that could have been encrypted into c under different keys. The *plausibility distance* is the minimum amount of information (in symbols) that the attacker needs to know so as to single out the real plaintext among $m_1, m_2, \ldots$.

We model the background information as a partitioning of messages into sets that are plausible as plaintexts, or implausible (for example, if the background information tells that the plaintext is a natural language text, then all random strings are implausible, while natural language strings are plausible). We assume that both parties, the honest and the adversary know this distinction, i.e., can distinguish plausible from implausible messages. This is shared knowledge by both parties as shown in Fig. 4.

Definition 6 (Plausibility Distance): Let $\Omega = P \cup I$ be the set of all messages, partitioned into the disjoint sets of plausible messages in P, and implausible messages in I. Let the message source be a random variable $Z \sim F(\Omega)$ with some distribution F over the set of all messages Ω, with entropy $H(Z)$. Let $H(Z|P)$ be the entropy conditional on that a random message is also plausible, and let $\mathbf{c}$ be a ciphertext encapsulating ℓ plaintexts sampled from the source Z. The *plausibility distance* of $\mathbf{c}$ is

$$N(\mathbf{c}) = \frac{\log_2 \ell}{H(Z) - H(Z|P)}$$

As with the unicity distance, the random cipher model can help here to approximate respective values; the only difference to the historic notion is that our division of meaningful and meaningless messages is here replaced by a distinction of plausible from implausible plaintexts, depending on background knowledge of the attacker.

Proposition 2: Let $\mathbf{c}$ be a False-Bottom ciphertext encapsulating ℓ messages sampled from some random variable Z. Among these, let us single out m^* as real, opposed to all other messages being for deception (fake). Let N_0 be the ciphertext's plausibility distance, according to Def. 6.

Then, if the adversary knows more than N_0 symbols of the real message m^*, it can unambiguously identify m^* from the list of recovered messages. In that case, the scheme is not deniable anymore.

Proof: Let Z be the random message source, and let $Z|P \sim F_P$ be the conditional random variable saying that the random message emitted by Z is also "plausible". The respective entropies are $H(Z)$ and $H(Z|P)$.

For a guess about which among $m_1, \ldots, m_\ell$ is Alice's actual secret m^*, we have

$$\text{total number of plaintexts} = 2^{H(Z)\cdot\ell} \tag{6}$$

$$\text{number of plausible plaintexts} = 2^{H(Z|P)\cdot\ell}, \tag{7}$$

The chances of a correct choice is thus

$$q = \frac{\text{good cases (7)}}{\text{all cases (6)}} = 2^{(H(Z|P)-H(Z))\cdot\ell} = 2^{-D\cdot\ell},$$

with the quantity D here playing the same role (but not being the same) as the language's redundancy in the historic notion of unicity distance.

In the random cipher model, the attacker would then know that among all keys, only one key gives the respective real plaintext m^*, so that if some candidate message m is meaningful, the given ciphertext admits $2^{\text{key-entropy}} - 1$ many incorrect trial decryptions. We have a likewise argument here: the attacker has recovered a set of keys, and is uncertain about which is the right key pointing to the real message m^*. Let us express this uncertainty as the message-entropy h, which is at most $h \leq \log_2 \ell$, for a total of ℓ plaintexts in the adversary's possession. The number of incorrect "keys" is 2^h, and the expected number of *incorrect* guesses about the real plaintext is $N_0 = (2^h - 1) \cdot q \leq 2^h \cdot q = 2^{h-n\cdot D}$. If $h - n \cdot D = 0$, then $N_0 \leq 1$, which brings down the incorrect guesses to approximately zero (similarly to what the original unicity distance means), which happens when we have more than

$$N_0 = \frac{h}{D} \leq \frac{\log_2 \ell}{D}, \tag{8}$$

symbols known as background knowledge. □

We stress that the converse implication does not follow from our arguments, meaning that the scheme is not necessarily plausibly deniable if the adversary knows less than N_0 symbols. If the attacker knows less than N_0 symbols about the real plaintext, it may still expect a unique and correct recovery of the real plaintext m^* from all recovered ones, based on its background knowledge. However, the scheme can remain *deniable*, since *there may exist* more than one meaningful possible plaintext. Nonetheless, we may be unable to convince the attacker to believe in a wrong plaintext; hence the deniability is *not plausible*. However, if we also let Alice draw her fake messages from the plausible set $P \subseteq \Omega$ (or if $\Omega = P$ directly), then we have $H(Z) = H(Z|P)$ and therefore $N_0 = \infty$. In that case, the scheme is plausibly deniable:

Corollary 1: Let Alice have sampled her real secret m^* from a source Z, and let her also have sampled the fake secrets from the same source Z, with a non-degenerate distribution. If Alice has added at least two messages to $\mathbf{c}$, then any brute-force decryption is plausibly deniable.

The practical difference between deniability and plausible deniability becomes apparent if we re-consider our initial setups of Alice storing information only for herself, or if Alice and Bob store information at a common location, e.g., a cloud. If it is just Alice storing data on her own computer, keeping decryption keys separate of it, then all messages m that go into the ciphertext vector (file) $\mathbf{c}$ come from Alice as a source, and hence have the same distribution. Hence, Alice can accomplish plausible deniability in the sense of Def. 5.

On the contrary, if Alice and Bob both access and add to the ciphertext (file) $\mathbf{c}$ stored at some common location like a shared cloud space, then there are messages from two (distinct) sources Z_{Alice} (for Alice) and X (for Bob or any other source) found in $\mathbf{c}$, and the joint random source is

$$Z = \begin{cases} Z_{\text{Alice}} & \text{with probability } p \text{ (data from Alice)} \\ X & \text{with probability } 1-p \text{ (not from Alice)} \end{cases}$$

if Alice has added a fraction of $0 \leq p \leq 1$ and Bob has added the residual fraction of $(1 - p)$ to the file $\mathbf{c}$. If the adversary then forces Alice to reveal what she stored in the cloud, she may just open one of Bob's data items and claim that it was hers. This fails if the adversary has sufficient background knowledge, i.e., more than $N_0 = \log \ell/(H(Z) - H(Z|\text{Alice}))$ symbols, to recognize Alice's messages based on its prior information. Here, conditioning on "Alice" is a shorthand notation to mean that the message came from Alice, in a slight abuse of notation. Let us work out the denominator in this expression explicitly: by definition of Shannon entropy, we have $H(Z) = pH(Z_{\text{Alice}}) + (1-p)H(X)$. By definition of conditional entropy, we have $H(Z|\text{Alice}) =$

$-\sum \Pr(Z, \text{Alice}) \cdot \log \Pr(Z|\text{Alice})$. From the conditional probabilities we find the joint probability $\Pr(Z, \text{Alice}) = \Pr(Z|\text{Alice}) \cdot \Pr(\text{Alice})$, in which $\Pr(Z|\text{Alice}) = \Pr(Z_{\text{Alice}})$, as the random variable Z is defined to be Z_{Alice} if Alice emits the message. Similarly, $\Pr(\text{Alice})$ is the chance for Alice to emit a message in first place, making $\Pr(Z, \text{Alice}) = p \cdot \Pr(Z_{\text{Alice}})$. Substituting these terms into the definition of conditional entropy, we find $H(Z|\text{Alice}) = -\sum p \cdot \Pr(Z_{\text{Alice}}) \cdot \log \Pr(Z_{\text{Alice}}) = p \cdot H(Z_{\text{Alice}})$, since p is a constant. Thus we have $H(Z) - H(Z|\text{Alice}) = H(Z_{\text{Alice}}) - p \cdot H(Z_{\text{Alice}}) = (1-p) \cdot H(Z_{\text{Alice}})$ The plausibility distance thus evaluates to $N_0 = \frac{\log \ell}{(1-p) \cdot H(Z_{\text{Alice}})}$.

Let us discuss the two extreme cases $p = 1$ or $H(Z_{\text{Alice}}) = 0$ for the last formula, either giving $N_0 = \infty$. If $p = 1$, then all messages in **c** were created by Alice, but as such, they all came from the same source, making the real ciphertext indistinguishable (statistically) from a ciphertext for a fake message. Plausible deniability thus holds. If $H(Z_{\text{Alice}}) = 0$, then the attacker has zero uncertainty about what messages were from sources other than Alice, and can filter out these from the list of plaintexts discovered. The remaining plaintexts, however, are again all from the same source (Alice) and it cannot tell them further apart to single out the "real plaintext".

The most general case is Alice having created plaintexts that are plausible or implausible for her, in which case our previous analysis would need to distinguish the conditional entropies into not only "coming from Alice", but instead "comes from Alice and is plausible" versus "comes from Alice and is implausible". This takes us back to the original setting of Alice putting only her own messages into the False-Bottom Encryption, at which she needs to take care that her fake messages should not be distinguishable from her real content. The bottom line is that:

- Background knowledge to distinguish messages by Alice from those by Bob will retain plausible deniability even in the extreme case, if Alice and Bob have both put more than one message into the encryption.
- To further narrow down Alice's real information to defeat her denial, more refined background information on Alice herself is needed, to tell her plausible messages apart from her fake messages.

VII. DISCUSSION AND CONCLUSION

We have shown a conceptually simple method of concealing information inside an existing sequence of strings, allowing for deceptive decryption in case of forced revelation of decryption keys. At a practical level, matters of storing or remembering the decryption keys have not been discussed, but the use of passwords, for example, is not difficult to imagine here: suppose that whenever we require random values to be chosen, we do so by invoking a pseudorandom number generator (PRNG, e.g., the standardized password-based key derivation function Argon2) that is seeded with a password that the user chooses. In that way, the storage of the value-pair list that constitutes the secret key sk_i for the message m_i, boils down to the choice of a password to open the message m_i, from which all random quantities in the process can be recomputed with the password as a seed for a PRNG. The implications to security are, in that sense, to be considered carefully, as the overall entropy about the secret reduces to the min-entropy of the password choice process that determines the hardness of guessing the password, which is the Shannon entropy.

Further generalizations may be the inclusion of a third party to establish a four-eyes principle in the opening of a message. That is, for example, one could substitute $\rho_{i,1}, \ldots, \rho_{i,n_i}$ by products $\rho_{i,1}^{(a)} \cdot \rho_{i,1}^{(b)} \ldots, \rho_{i,n_i}^{(a)} \cdot \rho_{i,n_i}^{(b)}$, with the individual factors coming from key-bases, respectively root-keys, that two persons, Alice and Bob, are given. In that case, an adversary forcing Alice to cooperate would also have to convince Bob to cooperate, in order to discover a meaningful message.

As yet another variant, note that the role of the factors from root key **r** and from **c** is "symmetric", and hence one could alternate the appending of parts to **c** with adding parts to the **r**. Storing **c** in a remote location and keeping **r** on one's own local computer then creates the seeming appeal of putting new information into **c** without actually letting **c** visibly grow. This instance of the scheme is, however, not considered as useful here, since it is nothing else than storing an encrypted version of a message locally, and letting the *key* to this message be stored remotely at a possibly untrusted location.

Practical room for improvement is in the scheme's necessity to remember and use all secret keys whenever there is a need to modify messages after they went into the ciphertext **c**. Abandoning this requirement, the key storage and management requirements fall back to those of a conventional secret key encryption with fixed key sizes. Hence, its "information-theoretic" security guarantees are in any case bounded by the size of the secret root key to be guessed, but the brute force complexity is still larger than just guessing the secret key, since the adversary may, except if there is so far only 1 message in **c**, still be uncertain about which parts of **c** may have been used to represent the secret message. Therefore, the concept of (plausible) deniability is the added value over brute-force attack resilience.

Based on our review of literature on deniable encryption schemes, the user can, in past schemes, come up with only one fake message as a counterpart to defend its secret. In contrast, our scheme allows us to bring more than one fake message to hide a secret. Also, this encryption scheme is editable in the sense that at any instant in time, we can update our message by changing only one element in the ciphertext or by changing the key indices. Furthermore, the scheme gives us the freedom to delete any message encrypted inside the ciphertext simply by replacing at least one element from the ciphertext with a random number. Deleting a message gives us the flexibility to prevent the user who was earlier accessing that message from doing it again. If the user wants to decrypt the ciphertext with the older key, the outcome will undoubtedly dissatisfy

him. Consequently, False-Bottom Encryption extends deniable encryption by the functionality of adding, editing and deleting possibly several plaintexts inside the ciphertext.

Our security definition does not account for adversaries profiling the access patterns of a user, which calls for additional techniques to either randomize or ''equalize'' all access sequences. Future work will thus investigate extensions to our scheme by means of private information retrieval or other techniques (see the related work, in particular [22]), to analyze if information-theoretic security remains accomplishable or deteriorates against attackers that profile the (physical) device usage.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for invaluable suggestions. Their input greatly improved the readability and quality of the text.

REFERENCES

[1] R. Canetti, U. Feige, O. Goldreich, and M. Naor, ''Adaptively secure multi-party computation,'' in *Proc. 28th Annu. ACM Symp. Theory Comput.*, Philadelphia, PA, USA, 1996, pp. 639–648. [Online]. Available: http://portal.acm.org/citation.cfm?doid=237814.238015

[2] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky, ''Deniable encryption,'' in *Proc. 17th Annu. Int. Cryptol. Conf.* (Lecture Notes in Computer Science), vol. 1294. Santa Barbara, CA, USA: Springer, 1997, pp. 90–104.

[3] A. Juels and T. Ristenpart, ''Honey encryption: Security beyond the brute-force bound,'' in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 8441, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, A. Kobsa, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, D. Terzopoulos, D. Tygar, G. Weikum, P. Q. Nguyen, and E. Oswald, Eds. Berlin, Germany: Springer, 2014, pp. 293–310, doi: 10.1007/978-3-642-55220-5_17.

[4] M. Durmuth and D. M. Freeman, ''Deniable encryption with negligible detection probability: An interactive construction,'' in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 6632, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and K. G. Paterson, Eds. Berlin, Germany: Springer, 2011, pp. 610–626, doi: 10.1007/978-3-642-20465-4_33.

[5] A. O'Neill, C. Peikert, and B. Waters, ''Bi-deniable public-key encryption,'' in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 6841, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. P. Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and P. Rogaway, Eds. Berlin, Germany: Springer, 2011, pp. 525–542, doi: 10.1007/978-3-642-22792-9_30.

[6] M. Klonowski, P. Kubiak, and M. Kutylowski, ''Practical deniable encryption,'' in *SOFSEM 2008: Theory and Practice of Computer Science* (Lecture Notes in Computer Science), V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, and M. Bieliková, Eds. Berlin, Germany: Springer, 2008, pp. 599–609.

[7] P. Gasti, G. Ateniese, and M. Blanton, ''Deniable cloud storage: sharing files via public-key deniability,'' in *Proc. 9th Annu. ACM Workshop Privacy Electron. Soc.*, Chicago, IL, USA, 2010, p. 31. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1866919.1866925

[8] M. H. Ibrahim, ''A method for obtaining deniable public-key encryption,'' *Int. J. Netw. Secur.*, vol. 8, no. 1, pp. 1–9, 2009.

[9] P. Chi and C. Lei, ''Audit-free cloud storage via deniable attribute-based encryption,'' *IEEE Trans. Cloud Comput.*, vol. 6, no. 2, pp. 414–427, Apr. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/7090980/

[10] R. Canetti, S. Park, and O. Poburinnaya, ''Fully deniable interactive encryption,'' in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 12170, D. Micciancio and T. Ristenpart, Eds. Cham, Switzerland: Springer, 2020, pp. 807–835, doi: 10.1007/978-3-030-56784-2_27.

[11] C. Dwork, M. Naor, and A. Sahai, ''Concurrent zero-knowledge,'' *J. ACM*, vol. 51, no. 6, p. 851–898, Nov. 2004, doi: 10.1145/1039488.1039489.

[12] M. Naor, ''Deniable ring authentication,'' in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 2442, G. Goos, J. Hartmanis, J. van Leeuwen, and M. Yung, Eds. Berlin, Germany: Springer, 2002, pp. 481–498, doi: 10.1007/3-540-45708-9_31.

[13] R. W. Zhu, D. S. Wong, and C. H. Lee, ''Cryptanalysis of a suite of deniable authentication protocols,'' *IEEE Commun. Lett.*, vol. 10, no. 6, pp. 504–506, Jun. 2006.

[14] A. Fiat and M. Naor, ''Broadcast encryption,'' in *Proc. Annu. Int. Cryptol. Conf.*, Jan. 1993, pp. 480–491.

[15] J. Li, Y. Wang, Y. Zhang, and J. Han, ''Full verifiability for outsourced decryption in attribute based encryption,'' *IEEE Trans. Services Comput.*, vol. 13, no. 3, pp. 478–487, May 2020. [Online]. Available: https://ieeexplore.ieee.org/document/7936626/

[16] J. Li, Q. Yu, and Y. Zhang, ''Hierarchical attribute based encryption with continuous leakage-resilience,'' *Inf. Sci.*, vol. 484, pp. 113–134, May 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0020025519300684

[17] J. Li, W. Yao, Y. Zhang, H. Qian, and J. Han, ''Flexible and fine-grained attribute-based data storage in cloud computing,'' *IEEE Trans. Services Comput.*, vol. 10, no. 5, pp. 785–796, Sep. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7390098/

[18] S. Reddy, P. S. Reddy, and P. Sravanthi, ''Audit free cloud storage via deniable attribute base encryption for protecting user privacy,'' *Int. J. Sci. Eng. Technol. Res.*, vol. 5, no. 17, pp. 3449–3451, 2016.

[19] R. Bassous, R. Bassous, H. Fu, and Y. Zhu, ''Ambiguous multi-symmetric cryptography,'' in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 7394–7399.

[20] A. Chakraborti, C. Chen, and R. Sion, ''POSTER: DataLair: A storage block device with plausible deniability,'' in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1757–1759. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2989061

[21] C. Chen, A. Chakraborti, and R. Sion, ''PD-DM: An efficient locality-preserving block device mapper with plausible deniability,'' *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 1, pp. 153–171, Jan. 2019. [Online]. Available: https://petsymposium.org/popets/2019/popets-2019-0009.php

[22] C. Chen, X. Liang, B. Carbunar, and R. Sion, ''SoK: Plausibly deniable storage,'' *Proc. Privacy Enhancing Technol.*, vol. 2022, no. 2, pp. 132–151, Apr. 2022. [Online]. Available: https://petsymposium.org/popets/2022/popets-2022-0039.php

[23] E.-O. Blass, T. Mayberry, G. Noubir, and K. Onarlioglu, ''Toward robust hidden volumes using write-only oblivious RAM,'' in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Nov. 2014, pp. 203–214, doi: 10.1145/2660267.2660313.

[24] C. Hargreaves and H. Chivers, ''Detecting hidden encrypted volumes,'' in *Communications and Multimedia Security*, B. De Decker and I. Schaumuller-Bichl, Eds. Berlin, Germany: Springer, 2010, pp. 233–244.

[25] J. Vera-del-Campo, J. Pegueroles, J. Hernández-Serrano, and M. Soriano, ''DocCloud: A document recommender system on cloud computing with plausible deniability,'' *Inf. Sci.*, vol. 258, pp. 387–402, Feb. 2014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0020025513002958

[26] D. Sidi and J. Bambauer, ''Plausible deniability,'' in *Privacy in Statistical Databases* (Lecture Notes in Computer Science), vol. 12276, J. Domingo-Ferrer and K. Muralidhar, Eds. Cham, Switzerland: Springer, 2020, pp. 91–105, doi: 10.1007/978-3-030-57521-2_7.

[27] C. E. Shannon, ''Communication theory of secrecy systems,'' *Bell Syst. Tech. J.*, vol. 28, no. 4, pp. 656–715, Oct. 1949. [Online]. Available: https://ieeexplore.ieee.org/document/6769090

[28] O. Goldreich, *Foundations of Cryptography*, vol. 2. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[29] A. Shamir, ''How to share a secret,'' *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979, doi: 10.1145/359168.359176.

[30] G. R. Blakley, ''Safeguarding cryptographic keys,'' in *Proc. Int. Workshop Manag. Requirements Knowl. (MARK)*, Jun. 1979, pp. 313–318, doi: 10.1109/MARK.1979.8817296.

[31] M. S. Wamser, S. Rass, and P. Schartner, "Oblivious lookup-tables," *Tatra Mountains Math. Publications*, vol. 67, no. 1, pp. 191–203, Sep. 2016. [Online]. Available: https://www.sciendo.com/article/10.1515/tmmp-2016-0039
[32] A. J. Menezes, K. H. Rosen, P. C. V. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, May 2020.

SHAHZAD AHMAD received the bachelor's degree in electronics engineering from the Harcourt Butler Technological Institute, Kanpur, India, in 2016, and the master's degree in electronics engineering with a communication and information systems specialization from Aligarh Muslim University (AMU), Aligarh, India, in 2021. He is currently pursuing the Ph.D. degree in computer science with the LIT Secure and Correct Systems Laboratory, Johannes Kepler University Linz, Austria. His research interests include plausible deniability, cloud security, and signal processing.

STEFAN RASS (Member, IEEE) received the degree in mathematics and computer science from Alpen-Adria-Universität Klagenfurt. He is currently a Full Professor with Johannes Kepler University Linz, Austria, where he is also a member of the LIT Secure and Correct Systems Laboratory. He participated in various nationally and internationally funded research projects, as well as being a contributing researcher in many EU projects and offering consultancy services to the industry. He has authored numerous papers related to practical security, security infrastructures, robot security, applied statistics, and decision theory in security. His research interests include decision theory and game-theory with applications in system security, especially robotics security, and complexity theory, statistics, and information-theoretic security.

PETER SCHARTNER received the master's degree in telematics from the Technical University of Graz, in 1997, with a focus on information security, and the Ph.D. degree in computer science from Alpen-Adria-Universität Klagenfurt, in 2001, with a focus on security tokens. He is currently an Associate Professor with the System Security Research Group, Alpen-Adria-Universität Klagenfurt, teaching courses on theoretical computer science, algorithms, and data structures, security, and cryptography, and a Lecturer with the Trier University of Applied Sciences. He participated in various nationally and internationally funded research projects. His research interests include applied system security, key management, security infrastructures, and the applications for security tokens, especially smartcards.