Nakula: Coercion Resistant Data Storage against Time-Limited Adversary

Hayyu Imanda
hayyu.imanda@cs.ox.ac.uk
University of Oxford
Oxford, United Kingdom

Kasper Rasmussen
kasper.rasmussen@cs.ox.ac.uk
University of Oxford
Oxford, United Kingdom

ABSTRACT

Both private citizens and professionals including journalists and whistleblowers can find themselves in a situation where they need to physically carry confidential data on a mobile device, through a situation where they might have their device seized and be subject to interrogation. In that case the user may be required to hand over the data by providing the password to unlock the device, violating confidentiality. Many existing proposals to address this issue involve the user lying to the interrogator to convince them that there is no data present, or that they forgot the password, or provide them with a second password that reveal different information. Although data hiding or alternative passwords can be useful solutions, we want to avoid this and instead focus on a scheme where the user can show that they cannot possibly access the data.

In this paper we propose Nakula, a mechanism that enables a user to lock down data with a single click (or voice command, gesture, etc.), enabling secure data transport. The information remains confidential against a very strong adversary who has full control over both the network and the device; and has the ability to force the user to cooperate through coercion. Nakula is designed so that the user does not have to lie or provide any misleading information at all. To achieve this, the user temporarily loses the ability to access the data and will need a trusted third party to recover it. We present a detailed design and security analysis of Nakula, and a proof-of-concept implementation that demonstrates the feasibility of using standard mobile phones to carry data. Finally we discuss several context-specific authentication methods that can be used with the scheme to enable data recovery in a variety of situations.

CCS CONCEPTS

• **Security and privacy → Security services**; Privacy protections; *Security protocols*; *Mobile and wireless security*; *Database and storage security*.

KEYWORDS

coercion resistance, secure data storage, strong adversary model, confidentiality

ACM Reference Format:
Hayyu Imanda and Kasper Rasmussen. 2023. Nakula: Coercion Resistant Data Storage against Time-Limited Adversary. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29–September 01, 2023, Benevento, Italy*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3600160.3600175

1 INTRODUCTION

A mobile phone contains an enormous amount of private information. This affects all individuals, however some of these information may be classed as more sensitive than others, for example: a list of sources for journalists, confidential data of organisations, state-level secrets, a list of passwords, or medical information. In turn, exactly for the reason that an insurmountable amount of data is accessible, it has not been uncommon that device searches—for personal devices or otherwise—happen by law enforcement officials or border agencies [17, 30].

Some countries have legislations in place that would allow state organisations the power to access electronic devices and demand the owner of the password to provide the password or unlock these devices [9, 18, 31]. In addition, it has also been reported that US government officials have copied the contents of digital devices that have been searched at the border, which is stored in a database for up to 15 years [15]. In worse situations, some parties with malicious intent have been known to coerce an authentication information to obtain monetary gain or access to confidential data [7, 16].

These situations illustrate the sheer importance of secure data transport, where there exists an entity who wish to break confidentiality. In these cases, the 'adversary' may have the (legal) capability to request, or even demand, the device user of their authentication information, in order to obtain access to the plaintext of all the data on the device. Previous attempts at solving this exact problem have involved, for example, full disk encryption [33]. However when met with this situation, the user when demanded the decryption password have to either provide the password or decline–the latter may be met with an undesired situation.

Other approaches to this problem may include *hidden volumes*, a method to hide data on a device so it is not apparent to others who look at it. For example, the device may be split up into different volumes, which unlocks depending on which password the user enters. The idea of the system is as follows: when the user is requested to enter their password, they will input a 'decoy' password instead, to which the file system will direct the user to a file directory with predetermined inconspicuous-looking data [33, 38].

With these approaches in mind, unless the user wishes to decline to provide the password, the user has two of options: provide the password and lose confidentiality, or alternatively hide the data and/or provide false information. The latter may indeed protect the data, however such practices are in direct conflict with the statement by the Electronic Frontier Foundation, who recommends against methods that deceive or mislead border agents about what data is present on a device [8]. In addition, an article published in the American Bar Association states that *"under no circumstances should you lie to a border agent or seek physically to interfere with the agent's performance of official duties"* [12]. We believe our suggested mechanism fits within these legal recommendations.

In this paper, we propose Nakula[1], an approach to protecting device data in the presence of a strong adversary that reflects the scenarios described. Specifically, we consider a situation which some data is required to be transported via a physical medium, when there exists a finite period of time in which the adversary has physical access to the device as well as coercive capability: that is, the adversary is able to demand the user carrying the device to reveal and provide any authentication information (e.g., passwords or biometrics). In addition, the adversary would have access to the user's communications channels, which is reflective of the capabilities of a state-level adversary.

We introduce the most important feature of Nakula: our system does not require any individual to lie or provide any false information even when coerced. We do not rely on hidden volumes or any data hiding mechanisms that are designed to mislead the adversary that some data doesn't exist. For this exact reason, previous solutions mentioned above are not suitable. Our approach comes from a simple concept: the user is unable to provide the information to the adversary because the user legitimately does not have access to it. The data, during the period of adversary control, is not available even to the user, because the user does not hold the key required to decrypt the encrypted data. This key is held remotely by a trusted third party, who will not return the necessary building blocks to recover the data, unless they are convinced that the user is legitimately requesting so, which is done after the adversary is no longer present on the device.

We summarise our contributions as follows:

- We create an infrastructure for secure physical data transport, where confidentiality against a coercive adversary with full access of the device is guaranteed without the user having to lie or deceive the adversary. Within this infrastructure, we introduce a trusted third party who maintains recovery keys as well as verifying the user authentication after the adversary is no longer present.
- We design protocols between the user and the trusted third party to achieve these goals and provide a proof-of-concept implementation of the scheme.
- We discuss different options in which the user can authenticate themselves to the trusted third party in order to recover the data that was temporarily inaccessible.

2 RELATED WORK

2.1 Plausible Deniability and Coercion Attacks

Non-repudiation as a security goal directly contrast the notion of deniability, and many proposals have been made in the past to allow a user to deny that a particular cryptographic action has been made. For example, deniable encryption [6] allows the sender to find a different message m' and random choice r' such that $\text{enc}(m', r')$ appears the same as the original sent ciphertext c. Though this has been improved in recent years, this scheme is still far from practical especially for large messages.

On the protocol level, another breakthrough in deniability came with the proposal of Off-the-Record Messaging (OTR) [5] where authorship of a particular message can no longer be linked to the original sender. The trick to this mechanism is short-lived session keys, as well as using a MAC key that is shared with the receiver; with the MAC key later published, the original message can be also signed by the receiver or anyone else. In addition, session keys that are no longer used are deleted, allowing forward secrecy, but also deny authorship. Similarly, deniable authenticated encryption [23] allows the receiver of a particular message to arbitrarily create fake messages as if they were from the original sender, hence the sender can always deny their involvement. A recent work [25] has looked into how humans perceive these 'proofs' on a practical level, through a courtroom situation.

Attacks with adversaries who have coercive capabilities have been discussed in the literature, and are sometimes called *rubber-hose cryptanalysis*. Many attempts at maintaining confidentiality in this situation comes hand-in-hand with information hiding, which allows *plausible deniability*: a situation such that there "be no irrefutable evidence concerning a disputed event or action" [26]. For example, Veracrypt [33], a fork of the discontinued Truecrypt project [32], is an open source encryption software that comes with the feature that allows the user to create a hidden operating system whose existence should be impossible to prove, as it lies within an existing VeraCrypt volume, and any free space on a VeraCrypt volume is filled with random data when it is created. In addition, if coerced, the user can input password to the outer volume which reveals non-sensitive information. Thus, you will not have to decrypt or reveal the password for the hidden volume. Hidden volumes are an example of a steganographic file system [1], a storage mechanism designed to give the user a very high level of protection against being compelled to disclose its contents. This work is followed by other systems [2, 20, 29].

Other work on authenticating under duress include a distress pin [7], which includes a threat model of categorisation of coercion, and the Funkspiel scheme [16], with the latter assuming the user can alter the hardware of the device.

An alternative to authentication under duress have included a neuroscience-based authentication based on implicit learning [3]: the 'password' is planted to the user over a training period, and will be detected upon authentication; however, a user cannot explicitly explain what the password is. In [14], the authors proposed a method that uses measurements of skin conductance so that the key generated is different when the user is coerced. Note that these methods require a very specific system model different from ours.

The password manager OnePassword published a feature called Travel Mode [11], to protect ultra-sensitive data when someone crosses a border. A user can turn on travel mode in advance of travel, which removes (not hides) some stored passwords that were marked as sensitive. If they are asked to open their password manager, they proceed with authentication as normal but the border agents would not be able to tell that there are data that are missing. To recover

[1]The name *Nakula* comes from the Mahabharata tales, originally written in Sanskrit. In the mythology, Nakula is a character who is trusted in keeping secrets.

their removed passwords, the user can then authenticate themselves and turn off travel mode. Of course, this does not work with our adversary that can request or demand the user to turn off travel mode when the adversary is present.

The methods above are different from our solution, as we are not trying to deny a particular action has been taken through ambiguity; or worse, knowingly making an incorrect statement or entering an authentication secret that does not reflect the true content of the data they hold. Instead, we use a trusted third party who controls the user's access to the plaintext. Indeed, the idea of using a trusted third party to store an encryption key has been previously discussed on both an informal and an academic setting [19, 28], as a response to a capture or a device seizure scenario. In addition, [36] also uses an entity that issues a remote wipe command should the user report that a device is stolen or lost. The concept of using encryption without providing someone the key to deny access to the information is not new [35] and has been used maliciously in the form of malware (e.g., [27]) but in this paper we use this technique to our advantage.

2.2 Secure Deletion

In most systems, data deletion is not securely deleted by default. In this paper, we assume 'secure deletion', where data is made irrecoverable from a physical medium [24]. There have been many work dedicated to secure deletion: in [10], it is noted that cryptographic protocols forgetting information is usually assumed; in turn, they defined and constructed a *secure erasable memory implementation*, which turns any storage device into a storage device that can selectively forget. In [4], the authors used a block cipher to forget information rather than protect it. Many other schemes, as well as adversary models of device capture are discussed in [24]. In [38], the authors proposed a method to securely delete data when under coercion. This is done through a deletion password, which when entered, will verifiably delete the data within a special disk in the device. Again, the use of a decoy password is not desirable.

3 DESIGN

In this section, we describe the goal of Nakula and the design choices we made to reach such goals.

3.1 Design Goal

The main goal of this paper is to keep information confidential in the presence of a strong, coercive adversary. Clearly, to ensure confidentiality some sort of encryption is needed. Given that our adversary is able to demand authentication secrets from the user, existing methods such as a password-based symmetric key encryption (e.g. full-disk encryption), for example, is not suitable as the user will simply reveal the correct password and the plaintext will be available to the adversary. A second reason why full-disk encryption is unsuitable is the duration in which encryption process takes place: encrypting a large amount of data within one click requires a large amount of time. This is unideal, given that our user requires the data to be encrypted quickly before the adversary obtains access to the device.

Alternatives that would be consistent with the recommendations have that have been suggested include backing up these data over the cloud, and then deleting them on the device [34]. However, this takes significant planning: if you are in a place where there exists no or limited connectivity, this might not be an option. Equally, if the data is of large size and the adversary is expected to have access to the device within a short period of time, the upload process might not be completed in time. Though this would be suitable for some data transport, we would like to explore alternatives where data size or connectivity are not limitations.

To solve this, we utilise what is normally called on-the-fly encryption, where the data is automatically stored encrypted with a symmetric key k without user action [20]. When the file is loaded, it is automatically decrypted. Our method is simple: when the user wishes to make the data unavailable to the adversary, the symmetric key k is made unavailable by encrypting it with a public key pk, before k is deleted, so that the user will not have access to k and hence the plaintext.

Obviously, if the user has the corresponding secret key sk either on the device or derived from the user's knowledge, then this is the same case as earlier: the adversary will just coerce the user to reveal sk. Our approach is to involve a trusted third party (the *backend*), who holds sk and will only perform computations to recover the data when they are convinced that any request to do so is legitimately done by the user after the adversary is present. Given our reliance to the backend, we want our system to be secure even if the backend is later compromised, which we will discuss in Section 3.2.

To summarise, the goals of Nakula are as follows:

- Secure physical data transport where confidentiality during the presence of a coercive adversary is guaranteed.
- Quick, simple data lock: the user should be able to encrypt the data within a push of a button (or any equivalent command, including voice and gesture).
- The user does not have to state any incorrect or misleading information to the adversary.
- Long-term secrecy: should the backend be compromised at a later stage, the backend will not have access to the data stored on the user's device.

We note that though our approach is aimed at being consistent with the recommendation of not hiding or providing false information, we understand that we have not discussed further legal issues, the risk of device confiscation, or risk of harm. These are extremely complex matters that are dependent, at the very least, on jurisdictions and we note that these issues are out of scope for our paper; some guidance for those travelling with a device have been discussed in [12, 34].

3.2 System Model and Design

In this paper, we consider the following honest players:

(1) the *user*: the party who wishes to maintain confidentiality of some data.
(2) the *backend*: the trusted third party backend. We assume that the backend is an honest-but-curious party.

We assume that secure deletion on the user's device is possible. That is, when we say some data is *deleted*, we mean that it is irrecoverable from the device. We refer the reader to some methods in Section 2.2.

We assume that the user can predict when the adversary is present, or is given warning to when such occurrence will happen, which may be within a short amount of time. The user's goal is to maintain confidentiality when the adversary is present, and we assume that the user remains truthful when coerced by the adversary.

In this section onwards, we consider the *data* to be a collection of information in plaintext form, that the user wishes to remain confidential against the adversary. We consider five separate actions that the user performs in order for the system to correctly function:

(1) **Setup.** As the name suggests, this includes the user setting up the system on the device. This step is necessary for the user and backend to obtain the necessary keys. In practice, this may include the installation of an application on the user's device, and after the setup, any data that the user wishes to remain confidential is to be collected and stored encrypted through the application.
(2) **Data Acquisition**. Here, the user has access to the data and can continue acquiring information that will be later locked.
(3) **Data Lock.** The purpose of data lock is simple: the encryption key is deleted, so the data within the device is now inaccessible to anyone—this includes both the adversary and the user.
(4) **Key Recovery.** The user connects to the backend to obtain a (masked) key back, while performing authentication function f_{auth}.
(5) **Data Recovery.** The user obtains back data.

The data flow on the user device happens as follows, and is shown in Figure 1. Firstly, the user performs user setup to obtain the backend's public key pk, and we assume that the user is able to communicate to the backend during user setup (e.g., before leaving for a business trip). We then describe the period of time between user setup and data lock to be the *data acquisition* period. During data acquisition, the user has access to the data in plaintext form, and is able to make changes to them. However, the data is always stored in encrypted form on the device and decrypted when it is loaded for the user to perform operations—the encryption and decryption of the data happens without user intervention. We denote the symmetric keys that store the data as *session keys*.

Before the adversary is present, the user can instigate data lock: at this point, the user loses access to the data. The user can perform data lock at any point after data acquisition, and multiple times before adversary access. That is, the user can perform data lock using a session key k_i and start a new data acquisition session with a new session key k_{i+1}. This allows the user to manage the risks of adversary presence better: though our system assumes that the user is given warning when the adversary will be present, in reality the there might be cases where this is not possible, for example when the phone is lost or taken with force. Equally, the user might simply wish to lock the data at the end of a working day. In that case, the user can minimise the amount of data that is accessible at any point by performing multiple data locks; we describe this in detail in Section 4.

As an extra layer of security, during data lock, before the user encrypts with pk to make the data unavailable, the symmetric key k is first 'masked' by encrypting k with a symmetric *masking key* k_R. The masking key is kept within the user's device, and this makes sure that the backend after decrypting will only have access to $\text{enc}_{k_R}(k)$, and never has direct access to k as an extra layer of protection. Specifically, during data lock, each session key k is masked with k_R, before being encrypted with pk. Then, the user deletes k and $\text{enc}_{k_R}(k)$. Note that we use public key encryption, instead of symmetric key encryption to ensure that the user at no point in the data life cycle has access to the secret key sk. Note that we can consider pk as a *deletion* key, and sk as a *recovery* key.

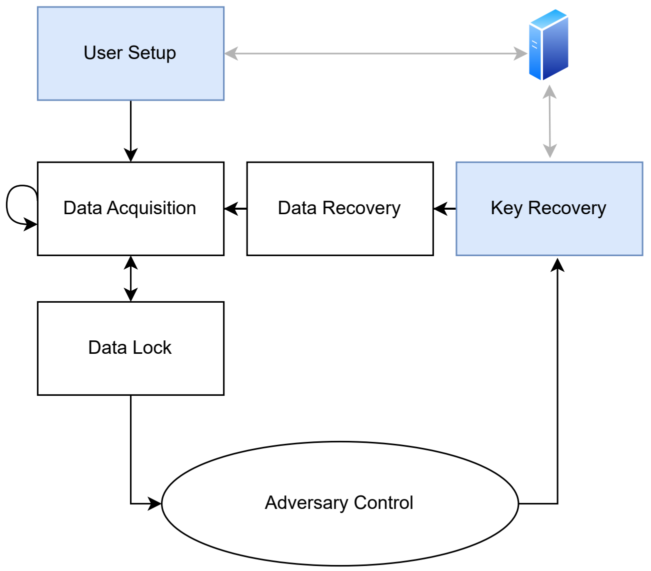

Figure 1: Nakula Design: After user setup, the user can collect data during data acquisition period; the user may lock the acquired data at any point, while still being able to collect additional data. Before adversary control, the user completed Data Lock and at this point the user do not have access to the data anymore. When the adversary is no longer present, the user can complete key recovery followed by data recovery to obtain the data back. The cycle starts again with data acquisition. During user setup and key recovery, the user communicates with the backend.

After data lock is instigated, the user enters the *adversary control* period, where the user no longer has access to the data as the encryption key is no longer accessible. The adversary control period is indeed the period of time in which we can guarantee confidentiality of the data when the adversary is present.

Note that because the user does not have access to, and has never had any previous knowledge of sk, the user is legitimately unable to decrypt the the encrypted data as there is no decryption key on the device to decrypt the data. In this way, if the user is demanded by the adversary the authentication information for the encrypted data, the user simply states that they are not able to decrypt (or *unlock*) the data as they do not have access to the decryption key. This can be supported by a well-designed UI on the device that states this information and how it works, but a large adoption of the system improves the probability that the adversary is aware of this mechanism, hence convinced that the user is telling the truth and reduces the risk of harmful coercion.

It is also important to note that when the adversary is present, the user does not have access to not only the session keys, but also the data. That is, the user will not be able to see or modify what is inside it, so it is important that the user is aware of this risk before using the mechanism.

When the adversary is no longer present on the device, the user can instigate *key recovery*, and we assume that communication with the backend is possible during this (e.g., when the user has left a region with limited connectivity). During this process, the user sends the ciphertext of the masked, encrypted session keys to the backend, to which the backend can use the recovery key to decrypt the ciphertext and obtain the masked key. This is then sent back to the user so that they are able to obtain the session keys and obtain access to the data through *data recovery*.

Of course, key recovery needs to be done in a way that preserves authentication of the user, even with the adversary having coercive capability and full, physical access to the device. The backend will not release the encrypted key (by decryption using sk) until they are convinced that any request to do so is not by the adversary, and is legitimately done by the user after the adversary is no longer present. Hence, the user should not have the ability to authenticate themselves relying solely on their own knowledge. For now, we assume a function f_{auth} that outputs a binary value such that $f_{\text{auth}} = 1$ if and only if the user is legitimately requesting key recovery without the influence of the adversary. We discuss this function in Section 7.

During key recovery, a new public key pk' is sent from the backend, with pk deleted on the user's device. This is because the adversary may store the device data, including the encrypted data and masking key for a long period of time. Should the backend be compromised at a later stage after key recovery, the adversary henceforth will not have access to the data.

Lastly, we stress that the backend is a trusted party. In practice, this can be the user's employer or trusted organisation and the implementation of Nakula, especially when it comes to designing f_{auth} (as we will discuss in Section 7) and a wide-enough adoption allows for the adversary to be aware of Nakula.

3.3 Adversary Model

We assume the adversary has full control over the network that the user operates on. During a finite time period (during *adversary control*), the adversary is able to coerce the user into revealing any authentication information and encryption keys, having full, physical access to the device and and its underlying logic. However, we do not assume the adversary is able to coerce the user to reveal the contents of the device that has been encrypted and made unavailable. We also assume that the adversary is able to clone the device, and store the cloned data for an indefinite amount of time (but not long enough to break the encryption algorithm used).

We acknowledge that the adversary can confiscate and destroy the device (or delete data on the device), but that is not something any protocol can prevent. Our functional guarantees only consider the case when the device remains in the user's possession.

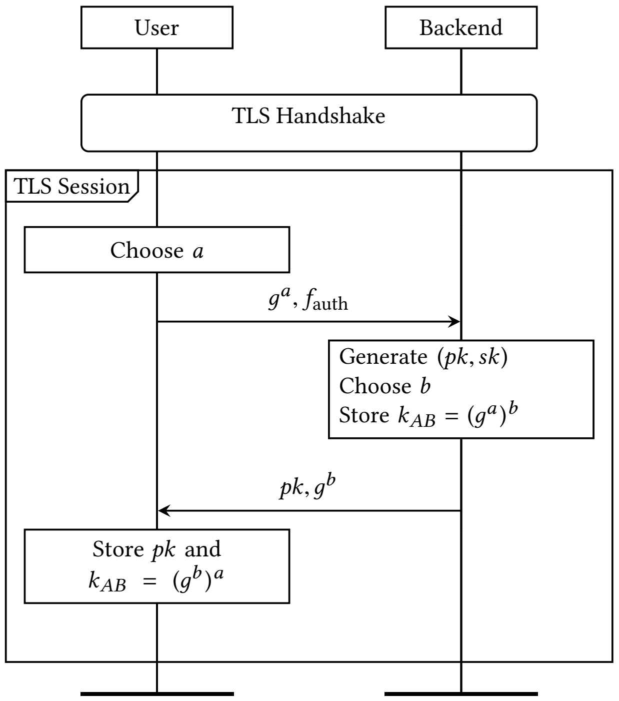

Figure 2: User setup protocol. In this protocol the user obtains the backend's public key pk and agree on a symmetric key k_{AB} with the backend. In addition, the user generates the masking key k_R associated to pk.

We assume the adversary is rational and is aware of the existence of Nakula and that it may be used in the system. In addition, we assume that the adversary does not have access to the backend's secrets during *adversary control*. The adversary's goal is to break confidentiality.

Note that our adversary model is consistent with what is usually considered for secure deletion: the adversary's goal is to recover deleted data objects after being given access to a physical medium that contained some representation of the data objects.

Lastly, we note that our use of the word *adversary* is simply from a systems point of view and not necessarily to imply that they have malicious intent.

4 NAKULA

In this section, we describe the inner workings of Nakula. First, we discuss the user setup protocol shown in Figure 2, where the user communicates with the backend to agree on the necessary keys. We then describe the life cycle of a session (Figure 3) and the various algorithms involved in Nakula (Algorithms 1, 2), ending with key recovery (Figure 4), where the user communicates with the backend again to obtain a masked session key and finally recover the original data (Algorithm 3).

4.1 User Setup

The goal of the user setup protocol is for the user to obtain the backend's public key pk and agree on a symmetric key k_{AB} used

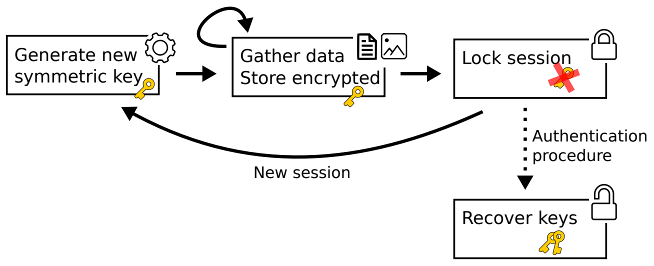

Figure 3: The user starts a new session by generating a new symmetric key according to Algorithm 1. The key is used to encrypt data as it is generated and the session is locked when the key is deleted. A new session can be started by creating a new key, or the user can recover locked sessions to regain access to locked data.

Algorithm 1: Session key generation

```
Function Generate($pk$):
    generate session key $k$ ;
    generate masking key $k_R$ ;
    store $\text{enc}_{pk}(\text{enc}_{k_R}(k))$, $k_R$, and $h(k)$ on disk ;
    return $k$
```

for subsequent communication with the backend. This protocol is an enrollment process that only has to happen once.

To initiate, the user connects to the backend through TLS, and verifies the TLS certificate. The user then chooses a Diffie-Hellman exponent a and sends over g^a to the backend within the authenticated TLS session, along with a preferred method of f_{auth}. Upon receipt, the backend generates a public-private key pair and chooses a Diffie-Hellman exponent b, and stores the shared key $k_{AB} = (g^a)^b$. The backend sends the public key pk and g^b to the user and the user computes $k_{AB} = (g^b)^a$. These are stored as pk will act as the user's deletion key for the current epoch, and k_{AB} will be used for subsequent communication with the backend.

Note that (pk, sk) is not global, and is generated uniquely for each enrolled user, and hence also acts as an identifier.

4.2 Session Start and Key Generation

After completing user setup, the user has obtained all the building blocks required to ensure that confidentiality during adversary control can be guaranteed assuming the user observes the proper life cycle of the data acquisition session. This life cycle is described in Figure 3.

The first step is to generate a session key k. This involves a couple of steps and is done with the `Generate` function shown in Algorithm 1. First the user generates the actual session key k and a masking key k_R. The masking key and the public key pk from the backend are then used to create an encrypted backup of k that the user cannot access without the assistance of the backend, since it is encrypted with pk. This allows the user to delete k at any time in order to protect the access to the data. The user also stores a hash of the key to be used for verification in case the session key ever needs to be recovered from the backend.

Algorithm 2: Session lock for active session (with key k)

```
Function Lock($pk$):
    secure delete session key $k$
```

4.3 Data Acquisition and Encryption

Now the user can start acquiring data (text, photos, etc.). It needs to be ensured that all data is encrypted with k. This is conceptually straightforward although there are some pitfalls if this is done on a mobile device.

Some mobile operating systems will cache images when they are taken, in order to improve the user experience when switching between applications. This is true to some extent for both Android and iOS and will need to be considered if every photo is sensitive. We discuss this in more detail in Section 6.

There is no practical limit on the length of a single session. The user can keep using the same session key k for as long as needed, but if there is ever a situation where the phone might be seized, e.g., going though a roadside checkpoint, or crossing an international border, the user can lock the session to make the data inaccessible.

4.4 Session Lock

Given that the data itself is already encrypted, and the user has already stored an encrypted version of k for data recovery, to lock a session all the user has to do is delete k as shown in Algorithm 2.

Once the session key k has been securely deleted, the data is not accessible to anyone, including the user. It is a key feature that the user cannot recover the data by themselves, because if they could, then they would be vulnerable to coercion by the adversary.

This works regardless of whether the adversary is aware of the technical details of Nakula, but a technically literate adversary plays to the user's advantage in practice, since they will understand that there is nothing the user can do, and therefore there is no point in continuing any interrogation.

We should note that if the user knows the data, e.g., they might remember a picture they took, Nakula does not protect against an adversary trying to extract such information from the user themselves. But at least the actual data in physical form, along with the possible intricate details, is beyond reach.

Note that the user may initiate session lock multiple times with the same public key pk (although with different session keys k_i), as shown in Figure 3. This allows the user to lock the session if there is any chance at all it might be needed, for example before going to bed in an unsecured hotel room. We denote by $\vec{C}_{pk}$ the list of ciphertexts under the public key pk, that is, $\vec{C}_{pk} = \{c_{b_1}, c_{b_2}, ...\}$.

To summarise, after session lock is completed i times, the user has stored on the device the following fragments, which we consider public information:

(1) $\text{enc}_{k_1}(\text{data}_1), \dots, \text{enc}_{k_i}(\text{data}_i)$
(2) $\vec{C}_{pk} = \{\text{enc}_{pk}(\text{enc}_{k_R}(k_1)), \dots, \text{enc}_{pk}(\text{enc}_{k_R}(k_i))\}$
(3) k_R
(4) $\vec{K} = \{h(k_1), h(k_2), \dots, h(k_i)\}$
(5) pk
(6) k_{AB}.

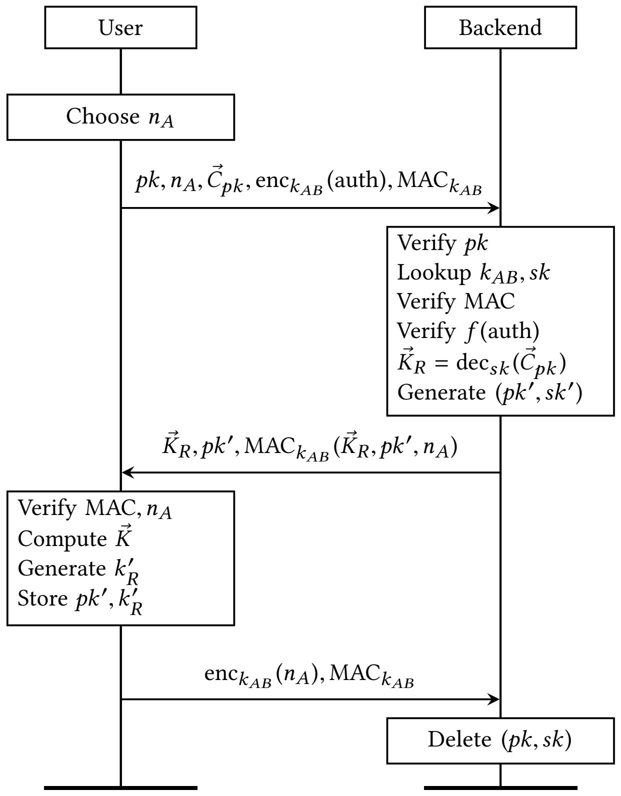
User
Backend
Choose n_A
$pk, n_A, \vec{C}_{pk}, \mathrm{enc}_{k_{AB}}(\mathrm{auth}), \mathrm{MAC}_{k_{AB}}$
Verify pk
Lookup k_{AB}, sk
Verify MAC
Verify $f(\mathrm{auth})$
$\vec{K}_R = \mathrm{dec}_{sk}(\vec{C}_{pk})$
Generate (pk', sk')
$\vec{K}_R, pk', \mathrm{MAC}_{k_{AB}}(\vec{K}_R, pk', n_A)$
Verify MAC, n_A
Compute $\vec{K}$
Generate k'_R
Store pk', k'_R
$\mathrm{enc}_{k_{AB}}(n_A), \mathrm{MAC}_{k_{AB}}$
Delete (pk, sk)

Figure 4: Session Key Recovery Protocol. The user supplies the encrypted (and masked) session key along with suitable authentication material (discussed in detail in Section 7), and gets back $\vec{K}_R$ from which the recovery key can be calculated, as well as a new public encryption key to use in the next epoch.

4.5 Key and Data Recovery

When the adversary is no longer present, the user can initiate key recovery, to recover access to the session key with the help of the backend. This is done with the Session Key Recovery Protocol shown in Figure 4.

The user chooses a nonce n_A and sends it along with the deletion key pk and a list of encrypted backup session keys $\vec{C}_{pk}$ to the backend. It is critical that the user cannot be forced by the adversary to complete this step. We discuss in Section 7 how this might be achieved in practice, but for the purpose of the protocol we model it as some authenticating information 'auth'.

The backend, upon receipt of the message from the user, verifies that pk is in the system, then looks up k_{AB} and sk which corresponds to pk. The backend then uses k_{AB} to verify the MAC, before verifying that $f_{\mathrm{auth}}(\mathrm{auth}) = 1$ (again, see Section 7), ensuring that the request from the user is genuine. The backend then decrypts $\vec{C}_{pk}$ with sk and sends this back to the user along with a new deletion key to be used for the next epoch.

The user verifies the $\mathrm{MAC}_{k_{AB}}$ and n_A to ensure integrity and freshness and then computes the set of session keys $\vec{K} = \{k_i\}$ by

Algorithm 3: Data Recovery for key k

```
**Function** Recover($k_R, pk, enc_{k_R}(k)$):
    $k = \mathrm{dec}_{k_R}(\mathrm{enc}_{k_R}(k))$ ;
    $\mathrm{data} = \mathrm{dec}_k(\mathrm{enc}_k(\mathrm{data}))$ ;
    delete $pk, k, k_R$ ;
    **return** *data*
```

decrypting each member of the list with k_R. A new masking key k'_R is computed and saved along with the new deletion key pk'.

Finally the user sends a confirmation to the backend that then deletes the now defunct (pk, sk), and as a final step to recover the data, the user runs Algorithm 3.

5 SECURITY ANALYSIS

We make the following assumptions:

(CA) The certificate authority issuing certificates of each protocol party is honest.
(SD) Once an object is *deleted*, it is irrecoverable from the medium (*secure deletion*).
(SS) Every encryption algorithm used is semantically secure.
(H) The hash function used by each protocol party is preimage-resistant; that is, given $h(m)$, it is infeasible to find m.
(M) The MAC used by each protocol party is secure.
(N) The probability that an honest party picks the same nonce twice is negligible.
(A) $f_{\mathrm{auth}}(\mathrm{auth}) = 1$ if and only if auth is provided at will by the legitimate user.

Guarantee 1. *At the end of user enrolment, and before adversary control, the user and backend shares a key only known to them.*

Proof. Firstly, the backend is authenticated using TLS (Assumption CA). Given that the key exchange happen within the same TLS session, confidentiality and integrity follows. □

Note that the above guarantee only ensures secrecy of k_{AB} strictly *before* adversary control, as the adversary has the capability of full access to the device, in which they can obtain k_{AB} and is then considered public information.

Guarantee 2. *If key recovery is completed successfully, then the backend is certain that the message coming from the user indeed comes from the user and is fresh.*

Proof. For an adversary to claim that they are the user, they need to pass the backend's MAC verification. That is, they need to either forge or replay $\mathrm{MAC}_{k_{AB}}(\dots, n_A)$. The latter is not possible due to Assumption (N).

Now assume the adversary does not have access to k_{AB}. Then due to Assumption (M) the adversary also isn't able to forge MAC.

Now assume the adversary has access to k_{AB}. Then the adversary will pass the backend's MAC verification. However, due to Assumption (A), we have $f_{\mathrm{auth}}(\mathrm{auth}) \neq 1$ as this is not requested freely by the user, and the protocol terminates. □

Guarantee 3. *If key recovery is completed successfully, and that the adversary does not compromise the backend during* adversary control, *then the user is certain that the reply coming from the backend: (1) is a response to the recent request from the user, and (2) indeed comes from the backend.*

Proof. (1) is immediate from Assumption (N). To break (2), an adversary who wishes to imitate the backend will have to pass the MAC verification; they can do this by forgery or replay. The latter is impossible due to Assumption (N). For the former, if the adversary does not have access to k_{AB} then the adversary cannot forge MAC due to assumption (M). So now assume the adversary has access to k_{AB} (as this becomes public information after adversary control). In this case, the adversary does not have access to sk, the private key corresponding to pk. Hence, except for negligible probability, $h(\mathrm{dec}_{k_R}(\mathrm{dec}_{sk}(\vec{C}_{pk})) \neq h(\vec{K})$, so the protocol aborts. □

Guarantee 4. *Assuming the backend doesn't have access to k_R, the backend does not have access to the session keys k used in the user's device.*

Proof. The backend only receives $\mathrm{enc}_{pk}(\mathrm{enc}_{k_R}(k))$, and due to having access to sk they can compute $\mathrm{enc}_{k_R}(k)$. As the backend doesn't have access to k_R, the proof directly follows from (SS). □

Guarantee 5. *During adversary control, no-one, including the user, has access to data unless the backend is compromised strictly during adversary control.*

Proof. During adversary control, the data is only stored of the form $\mathrm{enc}_k(\mathrm{data})$ (Assumption SD), so to obtain data they require the symmetric encryption key k (Assumption SS). However, k is only stored in the form of $\mathrm{enc}_{pk}(\mathrm{enc}_{k_R}(k))$ (Assumption SD), and $h(k)$. Given Assumption (H), they cannot obtain k from $h(k)$, so to obtain k they require both the masking key k_R and the secret key sk.

The masking key is present in the user's device, so the user and the adversary have access to them. However, sk is only held by the backend. The backend decrypts $\mathrm{enc}_{pk}(\mathrm{enc}_{k_R}(k))$ if and only if the user freely requests key recovery, but this is outside of adversary control. □

Guarantee 6. *After key recovery is completed successfully, the adversary will not have access to data even if they are able to compromise the backend.*

Proof. Note that during adversary control, the adversary has access to the fragments that we consider public information. If key recovery is successfully completed, the backend has deleted (pk, sk) (Assumption SD) and computes a new (pk', sk'), so even if the adversary is able to compromise the backend, they will only have access to sk' and
$\mathrm{dec}_{sk'}(\mathrm{enc}_{pk}(\mathrm{enc}_{k_R}(k))) \neq \mathrm{enc}_{k_R}(k)$ except with negligible probability. □

We proceed with the main functionality guarantee.

Guarantee 7. *If data recovery is completed successfully, then the no-one, other than the user, is able to gain access to data.*

Proof. As described in Algorithm 4, if data recovery is completed successfully, then key recovery has been completed. The user obtains $\vec{K}_R = \mathrm{dec}_{sk}(\mathrm{enc}_{pk}(\mathrm{enc}(k_R)(k_i)) = \mathrm{enc}(k_R)(k_i)$. Now, the user has access to k_R, so the user simply decrypts each element of the list to obtain the keys k_i.

Now, Guarantee 6 states that an adversary will not have access to data, so now we only need to consider the backend. However, the (honest) backend will only be able to decrypt using sk if sk was not deleted, contradicting Assumption (SD). □

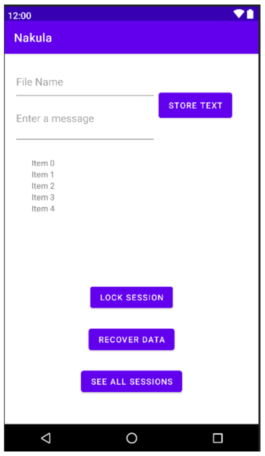

Figure 5: A screenshot of our Nakula app, which shows the UI for an unlocked session. In this app, the user can collect text information. When pressing "Store Text", the message is saved encrypted with a session key under the entered file name (e.g., Item 1). The user can still load the text until "Lock Session" is pressed, after which the user then is unable to have access to them. When the adversary is no longer present, the user can press "Recover Data" to obtain data back, provided that communication is possible and authentication has been completed.

6 IMPLEMENTATION

We implemented Nakula in the form of an Android application on the client-side, as shown in Figure 5, which was written using Android Studio [13]. We implemented the backend as a local web application using Flask [22]. The interaction between the app and the backend was tested locally using an Google Pixel XL emulator with Android API 33 on Android Studio.

In our implementation, the user conducts data acquisition from within the app. Should the user wish to store a certain message, they can enter it in a text box, along with an optional filename (which will be generated randomly if left blank) and stored within a specific folder on the device. Note that the file name will not be encrypted so the user should carefully choose the filename that will not reveal the content of the file.

For simplicity, we store these files within the app's internal storage; this makes sure that other applications on the device do not have access to these, however due to the fact that the data will be stored encrypted, in practice, confidentiality will still be maintained. Note that saving the file in internal storage adds the risk of data deletion, as data stored within the app's internal storage is automatically deleted when if the app is uninstalled.

We note that it is likely that cache of files are stored temporarily on the device memory, and hence during data lock it is necessary that these are cleared. We do not implement these as these are OS-specific.

Key management in Android has developed continuously. The most recent key management, the Android Keystore (for Android API Level ≥ 18) is designed so that keys cannot be extracted from the application process. This is rather challenging for us should we use the Android Keystore to generate and store our session keys, as we need to perform computations on it, and securely delete the key. In our solution, we generate the session keys outside of the keystore, and wrap each of them using an additional encryption key that is managed by Android KeyStore. As we generate the keys externally, we are able to access the key and compute the encryption with the masking key, followed by RSA encryption using the backend's public key. We don't specify the details of this in our protocol as it is implementation-specific.

Note that in our implementation, we are storing short text data just as a proof of concept. However, note that in a full-scale implementation this may include the storage of any other kind of data: images, videos, short and long text including passwords. In addition, the instigation of data lock can be changed to any input method: voice control, gestures, or a specific sequence of actions on the device.

Our proof-of-concept code is available at https://github.com/hayyuimanda/nakula.

7 AUTHENTICATION FUNCTION

So far, we've suggested that the security of the protocol reduces to the assumption that there exists an authentication function f_{auth} such that $f_{\text{auth}}(\text{auth}) = 1$ if and only if the user with authentication input auth is freely requesting key recovery, when the adversary is not present.

We have previously that a simple username and password authentication method does not suffice, as there is no mechanism to check whether or not the user is being coerced by the adversary during adversary control into entering those details.

Secondly, one may be tempted to design f_{auth} based on IP address or location; that is, the backend will only grant access if the user is in a country that is whitelisted (alternatively, not blacklisted) on the system through their IP address. However, authentication which relies purely on the device's claim of its location is also not desirable, as it is easy to spoof device location [37], even for a low-capability adversary.

We note that there are properties that f_{auth} are required to have:

(1) It cannot rely only on the user's knowledge, or the existence of an object on the user's device, as we are considering a coercive adversary who has access to the device.
(2) It is resistant to replay attacks. That is, the input requires some sort of timestamp.
(3) The process or the input to f_{auth} cannot be forged.

Though we plan to leave this open to the system designer, in this section we discuss several options that may be suitable depending on the context and the use-case. We summarise the advantages and disadvantages of the authentication methods we discuss in Table 1.

7.1 Physical Token

In this method, we consider the use of a separate physical device; the user's access to this device guarantees that the user is no longer within adversary control. In preparation of data gather and lock, the user is required to leave a physical token (e.g. a USB drive) at a certain location. Note that this token is required to have been set up by the backend, so that they share some authentication secrets; this is similar to a card reader as a security layer when the user wishes to authenticate themselves to their banks [21].

Because the security of f_{auth} directly reduces to access of the device, the location in which the user leaves the token needs to minimise the possibility of the adversary having access to it. For example, the user can leave the token in their home (assuming the adversary does not have access to it), their workplace, or handed to a trusted person, e.g. a lawyer (indeed, this is recommended in [28]). This choice is extremely important to ensure that only the user has this access.

Depending on the design of the physical token, the user, after leaving the space in which the adversary is present, simply requests key recovery with this authentication token plugged into the device, or provide a dynamic passcode generated by the device as part of auth. This can be combined with an extra layer of security, through a predetermined password.

The drawback of this method is that the user have to physically be in a predetermined location and is inflexible to users that are mobile; this means that the user cannot unlock and see the data until after the user is physically in such a position. This is also insufficient against adversaries that also have the extra capability of having access to these spaces (e.g. via collusion or through legal means).

7.2 Connection to a Local Network

This is a similar method to the use of a physical token, as this method requires the user to be physically present to a predetermined location, though with more flexibility: should Nakula be implemented as part of an organisation, there may be options that the organisation's network may be placed across different places. For example, a journalist may connect to the internal company network, which includes authenticating themselves. As the user is connected to this network, they can request key recovery to the backend with the authentication input being that the request comes from the user using this network to communicate.

Note that this has to be a local connection to the network, and not a remote connection, as otherwise the adversary will be able to exploit this. This is a more flexible method than a physical token, as there may be multiple locations in which the user can connect to, e.g. if the network is that of a multinational organisation which the user is an employee of.

However, this is again insufficient for the case which the adversary has access to this spaces, by legal means or otherwise.

7.3 Remote Authentication With Trusted Entity

This method requires a predetermined secret held by a trusted individual. For example, this can be an employee in the company who will be able to verify the user's face easily (e.g. a predetermined member of the same team/group).

To authenticate themselves, the user instigates a video call through a secure channel (e.g. an end-to-end encrypted video call software) to the trusted individual. The user then requests to the trusted individual that they would like to start key recovery. The trusted individual has to be convinced that the user is not within adversary control – it is up to this individual how that is judged, but they can request the user to show their surroundings, including the time of day outside and room structure. Note that if the adversary is able to convince the individual that the user is legitimately requesting this, then this method is far from secure, but depending on the situation, there are legal rights such that user should not be able to be forced make false statements to anyone.

To add another layer of security, and in a higher risk scenario, there may be be a predetermined secret which needs to happen between the two individuals, e.g. the user needing to make the call in front of a particular background or make certain gestures. Note that if this method is known to the adversary, then there is risk of the adversary being able to recreate the shared secret.

If the trusted individual is convinced that the user is freely requesting key recovery, then the trusted individual then provides auth to the user to forward to the backend as part of the key recovery request. The backend then proceeds with key recovery.

This method allows mobility as long as the user is able to connect remotely to the trusted individual, however this opens up the possibility of collusion between the adversary and the trusted individual, as well as possibility of human error by the trusted individual.

7.4 Time Delay

This is a possible authentication that does not require any external input as the previous three possibilities do. In this method, during user enrolment, the user specifies a time period between data lock and key recovery. If key recovery is completed before that time period ends, then the request will be denied. Should this method be used, there might be some additional information that would need to be included in the protocol: for example, the user needs to store $h(k_i, \text{time})$ for each session key. The time can be in the form of the closest hour to which data lock is instigated, and this is then passed on to the backend during key recovery. The backend will then loop through acceptable time period to decide whether or not to accept this.

This is useful for the case that there is a limit to adversary control (which there are many legal rights to), and is beneficial for users who are mobile and want access to their data from any physical space, as long as they can manage to not have access to that data during a specific time period.

8 CONCLUSION

We propose Nakula, a mechanism to allow secure data transport against a time-limited coercive adversary, which reflects a device seizure scenario during a border cross. We discussed why previous attempts to solve this are insufficient or undesirable, and Nakula is designed as an alternative that overcomes these limitations. We highlight that the main goal of Nakula is to allow data confidentiality when a coercive adversary is present on a device, with the goal being reached without the user having to lie or deceive the adversary, consistent with many legal recommendations of device seizure situations. Nakula is appropriate to use in situations where the user has to transport the data physically, and is useful in cases where the user is not able to, or does not wish to, transport the data over the internet (even in encrypted form)—for example, there may be poor internet connectivity for a large data transport, or when there is, they do not wish to increase suspicion by sending a large amount of data over the internet.

Table 1: Comparison of the authentication schemes mentioned in this section. We compare them on four factors: *Security* (how difficult is it to forge the authentication), *Usability* (how easy is it for the user), *Flexibility* (does it support changing plans), *Economy* (how cheap it is).

Mechanism	Security	Usability	Flexibility	Economy
Physical Token	High	Low	Low	High
Local Network	Med	Med	Low	Med
Trusted Entity	Med	High	High	Low
Time delay	Low	High	Med	High

To achieve the goal, we introduce the backend: a trusted third party who holds a secret key not accessible by the user; when the user instigates the system lock, the symmetric key that is used to encrypt the data is deleted, and only the backend is able to provide the building blocks necessary for the user to obtain the symmetric key back, which they will do if some authentication mechanism is completed by the user when the adversary is not present.

We discuss the design decisions for our scheme, proposed the protocols and algorithms within the architecture, and performed a security analysis on our protocols. We presented a proof-of-concept implementation and discussed the potential options for the authentication mechanism, which we do not specify due the context-specific scenario of the deployment of Nakula.

ACKNOWLEDGMENTS

The authors would like to thank Sebastian Köhler and Richard Baker in their support on the implementation of Nakula.

REFERENCES

[1] Ross Anderson, Roger Needham, and Adi Shamir. 1998. The Steganographic File System. In *Information Hiding*, David Aucsmith (Ed.). Vol. 1525. Springer Berlin Heidelberg, Berlin, Heidelberg, 73–82. https://doi.org/10.1007/3-540-49380-8_6 Series Title: Lecture Notes in Computer Science.

[2] Austen Barker, Staunton Sample, Yash Gupta, Anastasia McTaggart, Ethan L. Miller, and Darrell D. E. Long. 2019. Artifice: A Deniable Steganographic File System. In *9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/foci19/presentation/barker

[3] Hristo Bojinov, Daniel Sanchez, Paul Reber, Dan Boneh, and Patrick Lincoln. 2012. Neuroscience Meets Cryptography: Designing Crypto Primitives Secure Against Rubber Hose Attacks. In *21st USENIX Security Symposium (USENIX Security 12)*. USENIX Association, Bellevue, WA, 129–141. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/bojinov

[4] Dan Boneh and Richard J. Lipton. 1996. A Revocable Backup System. In *Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6* (San Jose, California) *(SSYM'96)*. USENIX Association, USA, 9.

[5] Nikita Borisov, Ian Goldberg, and Eric Brewer. 2004. Off-the-Record Communication, or, Why Not to Use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society* (Washington DC, USA) *(WPES '04)*. Association for Computing Machinery, New York, NY, USA, 77–84. https://doi.org/10.1145/1029179.1029200

[6] Rein Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. 1997. Deniable Encryption. In *Advances in Cryptology: CRYPTO '97*, Burton S. Kaliski (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 90–104.

[7] Jeremy Clark and Urs Hengartner. 2008. Panic Passwords: Authenticating under Duress. In *Proceedings of the 3rd Conference on Hot Topics in Security* (San Jose, CA) *(HOTSEC'08)*. USENIX Association, USA, Article 8, 6 pages.

[8] Sophia Cope, Amul Kalia, Seth Schoen, and Adam Schwartz. 2017. *Digital Privacy at the U.S. Border*. Technical Report. Electronic Frontier Foundation.

[9] US Customs and Border Protection. 2018. CBP Directive No 3340-049A. Subject: Border Search of Electronic Devices. https://www.cbp.gov/document/directives/cbp-directive-no-3340-049a-border-search-electronic-devices

[10] Giovanni Di Crescenzo, Niels Ferguson, Russell Impagliazzo, and Markus Jakobsson. 1999. How To Forget a Secret. In *STACS 99 (Lecture Notes in Computer Science)*, Christoph Meinel and Sophie Tison (Eds.). Springer, Berlin, Heidelberg, 500–509. https://doi.org/10.1007/3-540-49116-3_47

[11] Rick Fillion. 2017. *Introducing Travel Mode: Protect your data when crossing borders*. 1Password. https://blog.1password.com/introducing-travel-mode-protect-your-data-when-crossing-borders/

[12] Keith Fisher. 2020. *Update on Border Searches of Electronic Devices*. American Bar Association. https://www.americanbar.org/groups/business_law/publications/blt/2020/04/border-searches/

[13] Google for Developers. 2014. *Android Studio*. Google Developers. Retrieved March 16, 2023 from https://developer.android.com/studio

[14] Payas Gupta and Debin Gao. 2010. Fighting Coercion Attacks in Key Generation Using Skin Conductance. In *Proceedings of the 19th USENIX Conference on Security* (Washington, DC) *(USENIX Security'10)*. USENIX Association, USA, 30.

[15] Drew Harwell. 2022. *Customs officials have copied Americans' phone data at massive scale*. The Washington Post. https://www.washingtonpost.com/technology/2022/09/15/government-surveillance-database-dhs/

[16] Johan Høastad, Jakob Jonsson, Ari Juels, and Moti Yung. 2000. Funkspiel Schemes: An Alternative to Conventional Tamper Resistance. In *Proceedings of the 7th ACM Conference on Computer and Communications Security - CCS '00*. ACM Press, Athens, Greece, 125–133. https://doi.org/10.1145/352600.352619

[17] Rhett Jones. 2017. *Border Agent Demands NASA Scientist Unlock Phone Before Entering the Country*. Gizmodo. https://gizmodo.com/border-agent-demands-nasa-scientist-unlock-phone-before-1792275942

[18] Paul Karp. 2018. *Coalition's surveillance laws give police power to access electronic devices*. The Guardian. https://www.theguardian.com/australia-news/2018/aug/14/coalitions-surveillance-laws-give-police-power-to-access-electronic-devices

[19] Philip MacKenzie and Michael K. Reiter. 2003. Networked cryptographic devices resilient to capture. *International Journal of Information Security* 2, 1 (Nov. 2003), 1–20. https://doi.org/10.1007/s10207-003-0022-8

[20] Andrew D. McDonald and Markus G. Kuhn. 2000. StegFS: A Steganographic File System for Linux. In *Information Hiding*, Andreas Pfitzmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 463–477.

[21] NatWest. 2023. *What is a card reader and how do I use one?* NatWest. Retrieved 16 March 2023 from https://www.natwest.com/banking-with-natwest/how-to/card-reader.html

[22] Pallets. 2010. *Flask: web development, one drop at a time*. Pallets. Retrieved March 16, 2023 from https://flask.palletsprojects.com/en/2.2.x/

[23] Kasper Rasmussen and Paolo Gasti. 2018. Weak and Strong Deniable Authenticated Encryption: On Their Relationship and Applications. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE Computer Society, Belfast, 1–10. https://doi.org/10.1109/PST.2018.8514181

[24] Joel Reardon, David Basin, and Srdjan Capkun. 2013. SoK: Secure Data Deletion. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society, San Francisco, California, USA, 301–315. https://doi.org/10.1109/SP.2013.28

[25] Nathan Reitinger, Nathan Malkin, Omer Akgul, Michelle L. Mazurek, and Ian Miers. 2023. Is Cryptographic Deniability Sufficient? Non-Expert Perceptions of Deniability in Secure Messaging. In *2023 2023 IEEE Symposium on Security and Privacy (SP) (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 274–292. https://doi.org/10.1109/SP46215.2023.00095

[26] Michael Roe. 2010. *Cryptography and evidence*. Technical Report. University of Cambridge.

[27] M. Satheesh Kumar, Jalel Ben-Othman, and K.G. Srinivasagan. 2018. An Investigation on Wannacry Ransomware and its Detection. In *2018 IEEE Symposium on Computers and Communications (ISCC)* (25-28 June 2018). IEEE Computer Society, Natal, Brazil, 1–6. https://doi.org/10.1109/ISCC.2018.8538354

[28] Bruce Schneier. 2009. *Protect Your Laptop Data From Everyone, Even Yourself*. Wired. https://www.wired.com/2009/07/protect-your-laptop-data-from-everyone-even-yourself/

[29] Kian-Lee Tan, Hwee Hwa Pang, and Xuan Zhou. 2004. Hiding Data Accesses in Steganographic File System. In *Proceedings. 20th International Conference on Data Engineering*. IEEE Computer Society, Los Alamitos, CA, USA, 572. https://doi.org/10.1109/ICDE.2004.1320028

[30] Elise Thomas. 2018. *Sydney airport seizure of phone and laptop 'alarming', say privacy groups*. The Guardian. https://www.theguardian.com/world/2018/aug/25/sydney-airport-seizure-of-phone-and-laptop-alarming-say-privacy-groups

[31] Amar Toor. 2013. *UK border police can seize and download your phone's data for no reason at all*. The Verge. https://www.theverge.com/2013/7/15/4524208/uk-border-police-seize-download-mobile-phone-data-under-anti-terror-law

[32] Truecrypt. 2023. *Truecrypt*. TrueCrypt Foundation. https://truecrypt.sourceforge.net

[33] VeraCrypt. 2022. *Veracrypt*. IDRIX. https://www.veracrypt.fr/en/Hidden%20Volume.html

[34] Ben Wolford. 2018. *How to protect your phone or computer when crossing borders*. Proton. https://proton.me/blog/border-crossing-protect-electronics

[35] Adam Young and Moti Yung. 1996. Cryptovirology: Extortion-Based Security Threats and Countermeasures. In *Proceedings of the 1996 IEEE Conference on Security and Privacy* (Oakland, California) *(SP'96)*. IEEE Computer Society, USA, 129–140.

[36] Xingjie Yu, Zhan Wang, Kun Sun, Wen Tao Zhu, Neng Gao, and Jiwu Jing. 2014. Remotely wiping sensitive data on stolen smartphones. In *Proceedings of the 9th ACM symposium on Information, computer and communications security (ASIA CCS '14)*. Association for Computing Machinery, New York, NY, USA, 537–542. https://doi.org/10.1145/2590296.2590318

[37] Kexiong Curtis Zeng, Yuanchao Shu, Shinan Liu, Yanzhi Dou, and Yaling Yang. 2017. A Practical GPS Location Spoofing Attack in Road Navigation Scenario. In *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications* (Sonoma, CA, USA) *(HotMobile '17)*. Association for Computing Machinery, New York, NY, USA, 85–90. https://doi.org/10.1145/3032970.3032983

[38] Lianying Zhao and Mohammad Mannan. 2015. Gracewipe: Secure and Verifiable Deletion under Coercion. In *Proceedings of the 2015 Network and Distributed System Security*. Internet Society, San Diego, California, USA, 16 pages.