UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Dawn of the Golden Auditor
Constructing a Subversion-Resilient IND-CCA-Secure Encryption Scheme from Auditable Obfuscation

Dawn of the Golden Auditor
Konstruktion eines Subversions-Resilienten IND-CCA-Sicheren Verschlüsselungsverfahrens aus Auditable Obfuscation

Masterthesis

within the study courses of
IT-Security
at the University of Lübeck

handed in by
Jannik Westenfeld

issued and supervised by
Prof. Dr. Sebastian Berndt,
Prof. Dr. Thomas Eisenbarth

with the assistance of
M. Sc. Paula Arnold

Lübeck, February 21, 2025

Abstract

In recent years, Algorithm Substitution Attacks (ASAs) have become a hot topic of research in the field of cryptography. First introduced by Young and Yung in the late 1990s as *kleptography* and later coined ASA by Bellare, Paterson and Rogaway, the concept was widely deemed too artificial for real world considerations at first. However, revelations such as the publications from Edward Snowden [BBG13] showed that ASAs are not only a realistic attack vector, but could already be in use by malicious actors.

Cryptographic primitives, and especially encryption schemes, represent an attractive target for ASA exploitation. Attackers are able to circumvent breaking the schemes' security guarantees by introducing information-leaking side-channels into their implementations. As cryptographic implementations tend to be quite complex, it often is infeasible to prove that any particular implementation contains no subversion. Thus, a lot of research is conducted on adapting constructions in such a way that any possibly existing subversion can only be exploited if and only if this exploitation is also detectable. Such a construction is then labeled *subversion-resilient*.

While a lot of cryptographic primitives, such as IND-CPA-secure encryption, PRFs and signatures, have already been successfully adapted to resist ASAs, a subversion-resilient construction for IND-CCA-secure encryption has been elusive so far.

In this work, we provide such a subversion-resilient construction for IND-CCA-secure encryption from the novel concept of auditable obfuscation. Auditable obfuscation was first introduced by Banerjee and Galbraith in 2023 to model defenses against malicious modifications in obfuscation schemes. We analyze the model and show that auditable obfuscation ($\mathcal{AO}$) models an edge case of subversion-resilience., meaning that any auditable obfuscation scheme is inherently subversion-resilient and provides a new basis from which to source the property.

We further discuss a first application of this result, showing that we can leverage auditable obfuscation to achieve subversion-resilient indistinguishability obfuscation ($i\mathcal{O}$). Finally, we use the results from Sahai and Waters [SW14] on the construction of IND-CCA-secure encryption from $i\mathcal{O}$ and show that, by following its schemata using subversion-resilient building blocks, we can propagate this property to the resulting IND-CCA secure encryption scheme.

Zusammenfassung

In den letzten Jahren hat sich die Forschung zu Algorithm Substitution Attacks (ASAs) deutlich intensiviert. Das Themengebiet wurde in den späten 1990er Jahren von Young & Yung unter dem Begriff *Kleptography* begründet und später von Bellare, Paterson und Rogaway als ASA wieder aufgegriffen. Auch wenn das Setting erst als zu künstlich für reale Anwendungen angesehen wurde, so hat sich spätestens mit den Veröffentlichungen von Edward Snowden [BBG13] herausgestellt, dass es nicht nur einen realistischen Angriffsvektor darstellt, sondern womöglich bereits Anwendung findet.
Kryptographische Primitive, so wie Verschlüsselungsverfahren, stellen dabei ein besonders attraktives Angriffsziel dar, da Angreifer durch ASA eingeführte Seitenkanäle verwenden können um Sicherheitsgarantien der Verfahren zu umgehen. Da kryptographische Primitive meist sehr komplex in ihren Implementierungen sind, ist es oft unmöglich zu garantieren, dass eine spezifische Implementierung keine Subversion enthält. Forscher legen somit ihren Fokus auf das Adaptieren von Konstruktionen, bei denen eine Subversion nur dann ausnutzbar ist, wenn sie auch detektierbar ist. Solche Konstruktionen werden dann auch als *subversions-resilient* bezeichnet.
Für viele kryptographische Primitive, unter anderem IND-CPA-sichere Verschlüsselungen, PRFs und Signaturen, wurden schon subversions-resiliente Konstruktionen gefunden. Für IND-CCA-sichere Verschlüsselung konnte bisher jedoch noch kein Kandidat konstruiert werden. In dieser Arbeit stellen wir solch eine subversions-resiliente Konstruktion für IND-CCA-sichere Verschlüsselung vor.
Unsere Konstruktion baut dabei auf dem neuartigen Konzept von Auditable Obfuscation ($\mathcal{AO}$) auf. Auditable Obfuscation wurde von Banerjee & Galbraith in 2023 eingeführt, um Verteidigungsmaßnahmen gegen bösartige Obfuscation-Verfahren zu modellieren. Wir analysieren $\mathcal{AO}$ unter diesem Aspekt und zeigen, dass Auditable Obfuscation einen Randfall von Subversions-Resilienz darstellt. Komplexere subversions-resiliente Konstruktionen können somit auf $\mathcal{AO}$ aufbauen.
Wir diskutieren eine erste Anwendung, indem wir zeigen, dass man aus Auditable Obfuscation subversions-resiliente Indistinguishability Obfuscation bauen kann. Konstruktionen für IND-CCA-sichere Verschlüsselung aus Indistinguishability Obfuscation sind seit längerem bekannt, unter anderem die Konstruktion von Sahai and Waters [SW14]. Wir nutzen diese Konstrukion und zeigen, dass unter Verwendung von subversionsresilienten Bausteinen das resultierende Verschlüsselungsverfahren auch subversionsresilient ist.

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Jannik Westenfeld
Lübeck, 21. Februar 2025

Acknowledgements

Firstly, I would like to thank my advisor Sebastian Berndt, who not only made me aware of this thesis topic, but also supported me extensively throughout my research and writing phases. Secondly, I want to thank Paula Arnold and her invaluable comments throughout our meetings. Without Sebastian and Paulas combined advisory, this thesis would not have been written the way it is.
Further, I'd like to thank Thomas Eisenbarth for the possibility to realize this thesis as is, as well as my family for providing an environment in which I was able to fully concentrate on my work.
Lastly, I want to thank Eric Landthaler, Martin Weberpals and Timothy Imort for bouncing off ideas as well as proofreading this thesis.

Contents

1 Introduction

While historically, cryptographic research was singularly focused on the security of honest schemes, recent events, such as the Snowden revelations [BBG13], have shown that in real applications the honesty of implementations cannot be safely assumed. Parts of an implementation may have been *swapped out* against malicious counterfeits that *subvert* the output, leaving a hidden side-channel to exfiltrate information. Nowadays, such attacks are called Algorithm Substitution Attacks (ASA) [BPR14]. However, the attack scenario can be retraced to the late 1990s, when it was first described under the term *kleptography* [YY97].

ASAs generally work by mimicking the algorithm they subvert while adding some kind of side-channel to the resulting output. As an example, an ASA on a RSA implementation could work by modifying the probabilities with which any of its secret parameters p or q are chosen. An external party could then break RSA's security guarantees by only testing *high-probability* primes for p and q.

As ASA allow for the construction of adversary-controlled side-channels, they are often leveraged against *secure* algorithms to silently break security guarantees. As such, cryptographic primitives, and especially encryption schemes, are prime targets for such attacks. Additionally, due to the inherent complexity in most cryptographic primitives, it is often infeasible to prove whether any implementation was subverted or not [BPR14].

Research has thus been concentrated on hardening constructions against subversions rather than trying to prevent them [RTYZ17, AMV15, CMNV22]. The goal is to build constructions that, even if parts of them are subverted, either render these subversions detectable or, with overwhelming probability, unexploitable. Such constructions are then called *surveillance-resistant* [BPR14] or, more commonly, *subversion-resilient* [AMV15].

For a lot of cryptographic primitives, subversion-resilient constructions already exist. Some examples include IND-CPA-secure encryption [RTYZ17], EUF-secure signatures [AMV15, BBC24], MPC protocols [CMNV22] and PRFs [BBD+23]. However, there have been no successful attempts at a subversion-resilient IND-CCA-secure encryption scheme so far, neither for the symmetric nor public key scenario. IND-CCA-secure encryption can be understood as the *golden standard* of encryption schemes. Thus, it is of high interest to find a subversion-resilient construction for it. As standard constructions seemingly yield no results, it may be necessary to try and tackle this problem from a different direction.

Auditable obfuscation ($\mathcal{AO}$) is a comparatively new framework introduced in 2023 by Banerjee and Galbraith [BG23]. $\mathcal{AO}$ describes defense mechanisms against malicious obfuscation schemes. In real world applications, obfuscation schemes are mostly handled as black-box algorithms provided by a third party. However, prior to the work of Banerjee and Galbraith, this third party was generally assumed to be trustworthy, which may not always be the case. Auditable obfuscation then tries to adapt the obfuscation scheme in such a way that any code embedded by a malicious obfuscator is either detectable or non-exploitable.
When comparing the descriptions of $\mathcal{AO}$ with defenses against ASA, it is clear to see that, if not equivalent, both frameworks at least seem to handle similar ideas on different domains. It therefore is not far-fetched to suppose that $\mathcal{AO}$ schemes may be subversion-resilient, meaning subversion-resilient obfuscation could exist.
Constructing IND-CCA-secure encryption from obfuscation methods has been the topic of research before [SW14], however mostly based on indistinguishability obfuscation ($i\mathcal{O}$). Assuming $\mathcal{AO}$ is subversion-resilient, there may be a way to leverage its framework to construct subversion-resilient $i\mathcal{O}$ as well as subversion-resilient IND-CCA-secure encryption.

1.1 Contributions

We build upon the work of Banerjee and Galbraith in [BG23] and combine it with the findings of Berndt and Liśkiewicz from their 2017 paper "*Algorithm Substitution Attacks from a steganographic perspective*" [BL17] to provide a new look onto auditable obfuscation and its use cases. We do this by

- providing a succinct recap over auditable obfuscation in the Preliminaries,
- giving an intuition on the inverse problem to auditable obfuscation, i.e. malicious obfuscation, and its relation to algorithm substitution attacks,
- looking at malicious obfuscation from a *steganographic perspective*,
- showing how we can leverage auditable obfuscation to realize subversion-resilient indistinguishability obfuscation.

We further combine these results with the works of Sahai and Waters from their paper "*How to Use Indistinguishability Obfuscation*" [SW14], as well as the work of Bemmann et al. in [BBD+23], to construct a subversion-resilient IND-CCA-secure encryption scheme. We do this by

- providing a short introduction on how to achieve IND-CCA-secure encryption from indistinguishability obfuscation via the Sahai and Waters constructions,
- showing that cryptographic PRGs are inherently subversion-resilient,
- proving that the Goldreich-Goldwasser-Micali construction for PRFs [GGM86] is subversion-resilient,
- showing that subversion-resilient PPRFs, as necessitated by the Sahai and Waters construction for IND-CCA-secure encryption, are achievable,
- constructing a subversion-resilient IND-CCA-secure encryption scheme by combining the prior results.

We conclude by giving a short outlook on possible future work.

1.2 Related Works

In this thesis, we mostly work in both the obfuscation and the subversion-resilience setting. In this section, we discuss some related works from different fields of research.

Obfuscation

While obfuscation, and especially indistinguishability obfuscation, has been a hot topic of research in recent years, this thesis deals with a special type of obfuscation, namely auditable and subversion-resilient obfuscation. Auditable obfuscation has, to our knowledge, not been used outside of its introduction in [BG23], but earlier variants of related concepts like malicious obfuscation and verifiable obfuscation exist. Some examples are the works of Canetti and Varia in [CV09] and Badrinarayanan et al. in [BGJS16].
Canetti and Varia were the first to argue that malleability of obfuscation schemes could be used by a malicious party, laying the foundation for current research. The works of Badrinarayanan et al. built upon the works of Canetti and Varia, combining their theoretical results with later research into verifiable functional encryption. Their main results achieve defense mechanisms against *destructive* malicious behavior, auch as an adversary trying to render a ciphertext c indecipherable for verifiable functional encryption. Additionally, they also show that verifiable obfuscation is closely related to verifiable functional encryption in this specific context.
Further work was also done by Canetti et al. in [CCK+22], who, in addition to giving a more refined notion of obfuscation verifiability, built a *completely* CCA-secure encryption scheme from their notion of verifiable obfuscation. Similar to our work, they used the Sahai and Waters construction from $i\mathcal{O}$ [SW14] to achieve IND-CCA-security. Their goal was

to thereby extend the notions of non-malleability to further secure an encryption scheme against semantic adversaries. However, for their results, Canetti et al. rely on *honest* sub-procedures, making their results distinct from our works.
To our knowledge, there have been no previous attempts at linking auditable and verifiable obfuscation with subversion-resilience.

Algorithm Substitution Attacks and Subversion-Resilience

Similarly to obfuscation, algorithm substitution attacks and subversion-resilience have been the focus of recent publications. As an example, Berndt and Liśkiewicz showed in [BL17] that any black-box setting with enough entropy can be used to create ASAs. Building on this, future research focused more on concrete constructions. Some examples for specific cryptographic primitives include [CHY20], in which the authors construct algorithm substitution attacks on key encapsulation mechanisms (KEMs), and [AP22], where the authors construct ASA against receiver-focused cryptographic primitives.
For subversion-resilience, the research has been (mostly) split between two models, cryptographic reverse firewalls and watchdogs. For cryptographic firewalls, some of the achieved subversion-resilient cryptographic primitives include secure authenticated key exchanges (AKE) [LCW+25], secure password-authenticated key exchanges (PAKE) [CMMV25] and certain types of public key encryption [ZGL20, YLL+25]. However, none of the subversion-resilient encryption schemes have managed to achieve IND-CCA-security, distinguishing their results from ours.
Similarly, using watchdogs, it has been shown that subversion-resilient public key encryption [RTYZ17, BCJ21] and subversion-resilient authenticated encryption [BBD+23] are possible. However, these results again only achieve IND-CPA-security rather than IND-CCA-security. To our knowledge, there are no published works on subversion-resilient IND-CCA-secure encryption schemes.

2 Preliminaries

Throughout this thesis, we need to deal with a variety of different cryptographic settings: steganography, algorithm substitution attacks, obfuscation methods, pseudorandomness and encryption schemes. For most of these frameworks, notations and definitions have vastly differed throughout literature. Hence, we use this chapter to give a basic introduction into the primitives as we use them in this thesis. We also provide some context to their definitions, depict their security games, if applicable, and state the notations we will use from here onwards.
We start by providing a short introduction to the (*computationally*) adversarial setting and its notation. We then continue with the steganographic setting and algorithm substitution attacks, followed by a section on obfuscation. Lastly, we end the chapter with a section on pseudorandomness and encryption schemes.
A succinct overview of all commonly used notations can be found in Table 2.11 at the end of this chapter.

2.1 Adversarial Setting

When analyzing any kind of system, we need to set a general framework. For cryptography, two major frameworks have evolved over the years: the (*computationally*) asymptotic setting and the *(computationally)* concrete setting [KL14]. In the asymptotic setting, we assume that we can describe adversaries as randomized polytime-bounded algorithms that try to solve the problem of breaking any given cryptographic primitive Π. By polytime-bounded, we mean that the running time of the adversary is bounded by some polynomial evaluated on a security parameter λ. By showing that only an adversary running exponentially in λ can *effectively* break Π, we can then derive a computational security guarantee.
Contrary, in the concrete setting, we also describe adversaries as randomized algorithms, but fix their allotted time of execution not to a polynomial, but to an exact value. Similarly, we need to exactly describe the *effectiveness* of the adversaries as well. If no realistic adversary can break Π in their allotted time, a certain level of computational security can be derived based on the chosen parameters.
While the concrete setting is very useful for analyzing implementations in practice, it is not well equipped for a more theoretical analysis. As such, we will stick with the asymp-

totic adversarial setting throughout this thesis. We will now describe the basic notions used in the asymptotic framework, following the definitions from [KL14].
When working in the asymptotic setting, one of the most important notions is that of computational negligibility. We define a negligible function as follows.

Definition 2.1. Negligible Functions
A function f is *negligible* if for every polynomial $p(\cdot)$ there exists an N such that for all integers $n > N$ it holds that $f(n) < \frac{1}{p(n)}$. We denote an arbitrary negligible function in some security parameter λ with $\mathsf{negl}(\lambda)$.
For all negligible functions it further holds that:

1. $\sum\limits_{\mathsf{poly}(\lambda)} \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$
2. $\prod\limits_{\mathcal{O}(1)} \mathsf{poly}(\lambda) \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$.

Inversely, if for every polynomial $p(\cdot)$ there exists an N such that for all integers $n > N$ it holds that $f(n) > 1 - \frac{1}{p(n)}$, then f is *overwhelming*.

Generally, we use negligibility when dealing with success probabilities of adversaries. We model adversaries $\mathcal{A}$ as probabilistic polytime (PPT) turing machines, whose work we can describe with some polynomial $\mathsf{poly}(\lambda)$ in a security parameter λ. We then try and upper-bound their advantage in a function that is negligible in the same security parameter. By the second property of a negligible function, $\mathcal{A}$ thus is unable to succeed with overwhelming probability.
If $\mathcal{A}$ is not able to succeed with overwhelming probability, we deem a cryptographic primitive Π secure against that attacker. If Π is secure against all possible PPT adversaries, then we deem that primitive computationally secure. We formalize this with the following definition:

Definition 2.2. Computational Security
A cryptographic primitive Π is considered *computationally secure* for a security parameter λ in a security game game, if for all PPT adversaries $\mathcal{A}$ against Π in game it holds that their *advantage* $\mathsf{Adv}^{\mathsf{game}}_{\mathcal{A},\Pi}(\lambda)$ against game is at most negligible.
Formally, we require

$$\forall \mathcal{A} : \ \mathsf{Adv}^{\mathsf{game}}_{\mathcal{A},\Pi}(1^\lambda) \leq \mathsf{negl}(\lambda) \quad .$$

In both the description of negligibility and the definition of computational security, we used the attacker's advantage. However, we have not formally defined it so far. Generally, we say that the advantage of an attacker against some primitive Π is *how much better* this adversary is compared to the trivial attacker. Thus, we formally define the advantage as follows:

Definition 2.3. Advantage
Given a cryptographic primitive Π with a relating security game game and an adversary $\mathcal{A}$, the advantage of this adversary is defined as the absolute difference between the adversary's success probability and the trivial success probability of a simulator against game.
Formally, we write

$$\mathsf{Adv}^{\mathsf{game}}_{\mathcal{A},\Pi}(1^\lambda) = \left|\Pr[\mathsf{game}_{\mathcal{A}} = 1] - \Pr[\mathcal{S}^{\Pi}(1^\lambda)]\right| \quad ,$$

where $\mathcal{S}^{\Pi}(1^\lambda)$ describes a simulator with black-box oracle access to Π.

As can be derived from the definition of computational security, in analyzing the security of a primitive Π, we do not only want to bound the advantage of any individual adversary, but of all possible polytime adversaries. One possibility would be to analyze each adversary individually, but this is generally infeasible. Instead, we argue using the *best* adversary, meaning the $\mathcal{A}$ with the highest advantage. As an attack by this adversary constitutes the worst case against Π, we can describe the security in relation to it. For ease in notation, instead of defining a security notion, we define the concept of insecurity:

Definition 2.4. Insecurity
The insecurity of a cryptographic primitive Π is defined as the maximum advantage any adversary can have against Π.
Formally, we write

$$\mathsf{InSec}^{\mathsf{game}}_{\Pi}(\lambda) = \max_{\mathcal{A}} \mathsf{Adv}^{\mathsf{game}}_{\mathcal{A},\Pi}(\lambda) \quad .$$

This concludes all notations necessary for the adversarial setting. We now continue with definitions and notations for the steganographic and substitution settings.

2.2 Steganography, Algorithm Substitution Attacks & Subversion-Resilience

In this thesis, we will make use of side-channels embedded into the output of differing algorithms. The simplest case of this, secretly encoding some hidden message into some public message, is commonly known as *steganography*. Analysis of such schemes is defined in the steganographic setting [HvAL09]. In comparison, algorithm substitution attacks swap out honest algorithms in implementations for a malicious counterpart. This *subverted* algorithm is then used to facilitate a side-channel via which secrets can be leaked. Their analysis is defined in the substitution setting [BPR14, DFP15].
While the two settings seem slightly different, it was shown that algorithm substitution attacks equal steganography on a certain kind of channel [BL17]. As such, it is possible to translate between the terminology of both settings without loss of precision. However, while the definitions may map to each other, it is in specific circumstances more useful to use one setting over the other, i.e. to simplify proofs. As such, we will introduce both settings.
The use of steganography definitions will mostly be contained to Chapter 4, whereas the nomenclature of ASAs, and especially subversion-resilience, will be used throughout the entire thesis. As Chapter 4 follows a proof from [BL17] and its revised write-up in [Ber18] closely, we deemed it valuable to keep our definitions close as well. The definitions in this section are thus adapted from [Ber18] and mostly differ in symbolic notation.

2.2.1 Steganography

As mentioned above, in the steganographic setting, we handle the secret encoding of some hidden messages into public messages. While we could define the encoding of singular messages, it generally is assumed that we want to create a hidden communication channel. Communication channels originate from the field of information theory. As such, for our formal definition of channels, we will closely follow its definition.

Definition 2.5. Channel ($\mathcal{C}$)
Given an alphabet Σ, a channel $\mathcal{C}$ is a function that maps an element $\mathsf{h} \in \Sigma^*$, called the channel history, and a number $n \in \mathbb{N}$, called the document length, to a probability distribution on Σ^n, called the channel distribution. The elements of $\mathcal{C}$ are called *documents* or *messages* and are denoted by msg.
For a given history h and document length n the resulting channel distribution is denoted by $\mathcal{C}_{\mathsf{h},n}$. We will omit n if it is clear from context.

In defining channels, we used something called a channel history, which we only described as an element of Σ^*. As some of our proofs rely on the construction of a valid history, we will define it more formally.

Definition 2.6. Channel History
Given a channel $\mathcal{C}$, the associated channel history $\mathsf{h}_\mathcal{C}$ is defined at a timestamp $i \in \mathbb{N}$ as a stream of all messages set on $\mathcal{C}$ prior. For a given i, this means that

$$\mathsf{h}_\mathcal{C}(i) = (\mathsf{msg}_0, \mathsf{msg}_1, \ldots, \mathsf{msg}_{i-1}) \quad .$$

For timestamp 0, $\mathsf{h}_\mathcal{C}(0)$ is defined as the empty stream $\emptyset$.

With this, we formalized the underlying information theoretic principles of steganography. While we did informally describe the goal of the steganographic setting, we have not yet formalized an instantiation of the framework. A set of algorithms that tries to create a steganographic channel and then use it is most commonly called a stegosystem (StS). We formally define it as such:

Definition 2.7. Stegosystems (StS)
Let $\lambda \in \mathbb{N}$ be a security parameter and $\mathcal{C}$ a channel. A *(secret-key)* stegosystem StS is a triple of PPTMs $(\mathsf{StS.Gen}, \mathsf{StS.Enc}, \mathsf{StS.Dec})$, wherein

- StS.Gen is a *key-generation* algorithm taking the security parameter to produce its output.
$$\mathsf{StS.Gen}(1^\lambda) \to \mathsf{ak} \in \{0,1\}^{n(\lambda)}$$
- StS.Enc is an *encoder* algorithm taking a key ak, hidden message $\mathsf{sm} \in \{0,1\}^{\mathsf{poly}(\lambda)}$, history $\mathsf{h}_\mathcal{C}$ and some state information $\sigma \in \{0,1\}^*$ to produce a single output document $\widetilde{\mathsf{msg}} \in \{0,1\}^{\mathsf{poly}(\lambda)}$, which encodes some part of sm, as well as an updated state σ'.
$$\mathsf{StS.Enc}(\mathsf{ak}, \mathsf{sm}, \mathsf{h}_\mathcal{C}, \sigma) \to (\widetilde{\mathsf{msg}}, \sigma') \in \{0,1\}^{\mathsf{poly}(\lambda)} \times \{0,1\}^*$$
- StS.Dec is a *decoder* algorithm taking a key ak and a stream of $\mathsf{poly}(\lambda)$ many documents $(\mathsf{msg}_1, \ldots, \mathsf{msg}_{\mathsf{poly}(\lambda)})$ to produce a recovered message sm′.
$$\mathsf{StS.Dec}(\mathsf{ak}, (\mathsf{msg}_1, \ldots, \mathsf{msg}_{\mathsf{poly}(\lambda)})) \to \mathsf{sm}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$$

The security of a stegosystem is, similarly to other cryptographic primitives, defined via cryptographic games. For this thesis, we will focus on the StS-CHA-σ game also known as the stegosystem chosen-hiddentext-attack with state. A visual representation of this game can be found in Figure 2.1.
The game is played by a warden W on a channel $\mathcal{C}$, who needs to decide if there currently is a stegosystem StS running on $\mathcal{C}$ or not. If the advantage of W in this game is, at most, negligible for a given stegosystem StS, then StS is secure against that warden.

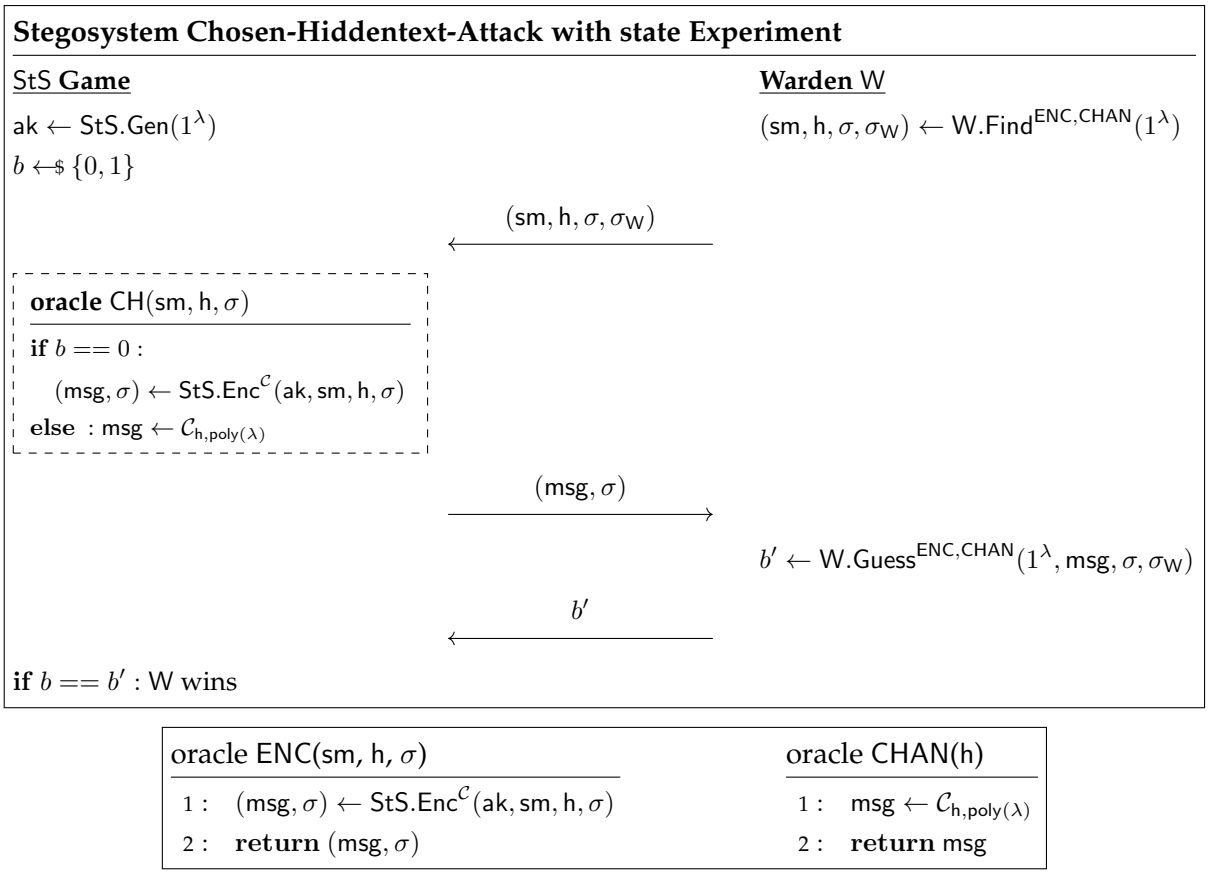


Figure 2.1: Depiction of the Stegosystem Chosen-Hiddentext-Attack with State Experiment, abbreviated to StS-CHA-σ. A warden W tries to determine if channel $\mathcal{C}$ contains a stegosystem StS. For this, they are given two oracles, ENC and CHAN, that provide them examples of honest messages and messages with an encoding. Using these oracles, W constructs a *best-case* for themselves, and queries the challenge oracle CH accordingly. Depending on the output of CH, the warden must then decide and wins if they guess correctly.

While security is an important metric for a stegosystem, it is not the only relevant property. Another such property of the stegosystem is that the encoded message can be reliably

decoded by the receiving party. Similarly to the insecurity of a cryptographic primitive, for ease in notation, we define its opposite, the unreliability of a stegosystem StS. The unreliability thus is defined via the worst-case encoding for StS. Its formal definition is as follows:

Definition 2.8. Unreliability
Given a channel $\mathcal{C}$ and related stegosystem StS, the unreliability $\mathsf{UnRel}_{\mathsf{StS},\mathcal{C}}(\lambda)$ is defined as the maximum probability of the decoder algorithm failing. Formally, we write:

$$\mathsf{UnRel}_{\mathsf{StS},\mathcal{C}}(\lambda) = \max_{\mathsf{ak},\mathsf{sm},\mathsf{h}} \Pr\left[\mathsf{StS.Dec}(\mathsf{ak}, \mathsf{StS.Enc}^{\mathcal{C}}(\mathsf{ak}, \mathsf{sm}, \mathsf{h})) \neq \mathsf{sm}\right]$$

where the probability is taken over the internal coin-flips of StS.Enc, StS.Dec and the samples of $\mathcal{C}$.

While we defined the unreliability for stegosystems only, we note that the idea behind it can easily be adapted to other adversarial primitives. With all necessary definitions for steganography introduced we move on to definitions for algorithm substitution attacks and subversion-resilience.

2.2.2 Algorithm Substitution Attacks

As mentioned before, algorithm substitution attacks define attacks against algorithmic implementations. An adversary $\mathcal{A}$ tries to introduce a subverted algorithm, the ASA, into an implementation with the goal of creating a side-channel through which secrets can be leaked. Originally, algorithm substitution attacks were known as kleptographic attacks, first described by Young and Yung in [YY97]. However, until the Snowden revelations in 2013 [BBG13], they were mostly disregarded as theoretical in nature.
In response to the revelations, an adapted version of the kleptographic setting, the algorithm substitution setting, was then constructed in [BPR14] and further refined in [DFP15]. The goal of the two works was to formalize security notions in a subversion setting, both modeling the adversary's, as well as the defender's, goals. Both papers define algorithm substitution attacks as a set of probabilistic polytime Turing machines (PPTMs) that work against a cryptographic primitive Π such that the ASA mimics the output of Π while encoding some secret message sm in the outputs. The secret message can then be retrieved from the output via an extraction algorithm.
The formal definition of an ASA over a symmetric encryption scheme SES is as follows:

Definition 2.9. Algorithm Substitution Attack (ASA)
Let $\lambda \in \mathbb{N}$ be a security parameter and SES a symmetric encryption scheme. An algorithm substitution attack ASA against SES then is a triple of PPTMs $(\mathsf{ASA.Gen}, \mathsf{ASA.Enc}, \mathsf{ASA.Ext})$, wherein

- ASA.Gen is a *key-generation* algorithm taking the security parameter to produce its output
$$\mathsf{ASA.Gen}(1^\lambda) \to \mathsf{ak} \in \{0,1\}^{n(\lambda)} \quad .$$
- ASA.Enc is a *subverted encryption* algorithm taking an adversary key ak, secret message $\mathsf{sm} \in \{0,1\}^{\mathsf{poly}(\lambda)}$, encryption key $\mathsf{k} \in \mathrm{Supp}\big(\mathsf{SES.Gen}(1^\lambda)\big)$, encryption message msg and a state σ to produce a ciphertext $c \in \{0,1\}^{\mathsf{poly}(\lambda)}$, which looks like an output from SES while also encrypting parts of sm, and an updated state σ'
$$\mathsf{ASA.Enc}(\mathsf{ak}, \mathsf{sm}, \mathsf{k}, \mathsf{msg}, \sigma) \to (c, \sigma') \in \{0,1\}^{\mathsf{poly}(\lambda)} \times \{0,1\}^* \quad .$$
- ASA.Ext is an *extraction* algorithm taking an adversary key ak and $\ell = \mathsf{poly}(\lambda)$ ciphertexts $(c_1, \ldots, c_\ell)$ to produce a recovered message sm′
$$\mathsf{ASA.Ext}(\mathsf{ak}, (c_1, \ldots, c_\ell)) \to \mathsf{sm}' \in \{0,1\}^{\mathsf{poly}(\lambda)} \quad .$$

While we do give the definition of an ASA over a symmetric encryption scheme, we can adapt the description to any cryptographic primitive Π. To adapt an ASA, we need to model ASA.Enc not as a subverted encryption algorithm, but rather as a subverted version of the output of Π. For uniform nomenclature, we decided to stick with the definition over a SES.
The *security* of an ASA is defined by the undetectability of its subversion. We model this security via a cryptographic game. In this thesis, we will use a chosen subversion setting in which a watchdog is allowed to specify the secret message that the subversion should embed via its introduced side-channel. If the advantage of the watchdog is negligible in this setting, the ASA is deemed secure. A visual representation of this game can be found in Figure 2.2.
Note the similarity between both the definition of ASA and StS, as well as their respective security games. The two settings seemingly map between each other, with the decoder algorithm of StS acting as the extraction algorithm in ASA and vice versa. This similarity is, as mentioned before, no coincidence. Berndt and Liśkiewicz proved in [BL17] that

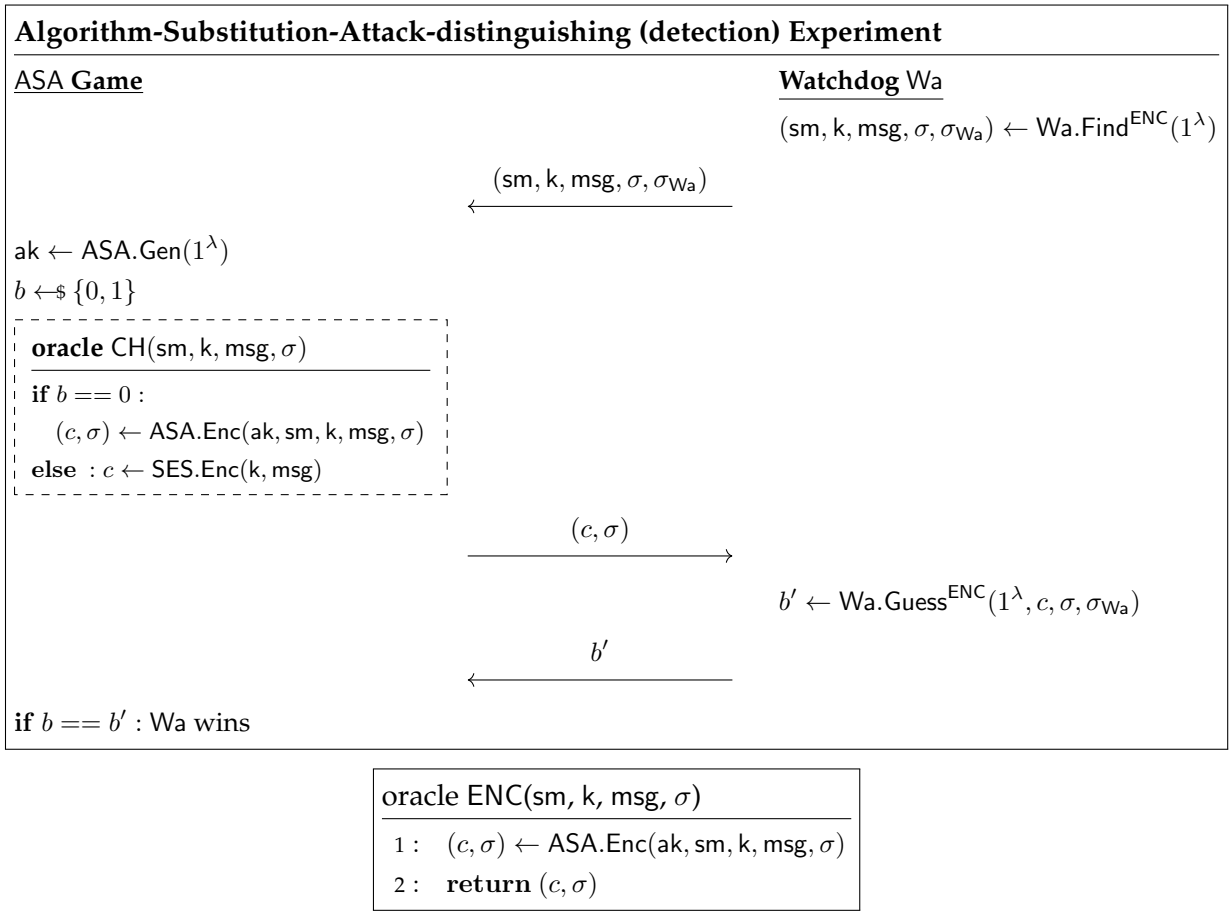


Figure 2.2: Depiction of the Algorithm-Substitution-Attack-detection experiment, abbreviated to ASA-Dist. A watchdog Wa tries to determine whether the output of a symmetric encryption scheme SES has been subverted or not. For this, they are give an encryption oracle, providing them with example subversions. Wa then uses this to create a *best case* distinguishing case, which they query on their challenge oracle CH accordingly. Depending on the output of CH, the watchdog must then decide and wins if they guess correctly.

the two settings are equivalent when working on a channel $\mathcal{C}$ built over the input-output behavior of a cryptographic primitive Π. If an implementation of Π could not be used for an ASA, then a channel over Π can not contain a stegosystem StS.
We describe the formal requirements for such an implementation in the follow section.

2.2.3 Subversion-Resilience

As mentioned in the last section, when describing algorithm substitution attacks, we also want to model implementations that defend against such attacks. The authors of [BPR14]

constructed a property called surveillance-resistance to achieve this goal. However, the property was improved on in [DFP15] under the new name of subversion-resistance, or more commonly refereed to as subversion-resilience. A subversion-resilient algorithm is modeled using watchdogs Wa that assess an implementation before it is used. We formally define the property of subversion-resilience as follows:

Definition 2.10. Subversion-Resilience
An algorithm Π is subversion-resilient if and only if an algorithm substitution attack ASA is detectable by a watchdog Wa, meaning

$$\forall \mathsf{ASA}\ \exists \mathsf{Wa}\ :\ \Pr\left[\mathsf{Wa.Guess}^{\mathsf{ENC}}(\mathsf{ASA.Enc}^{\Pi}(\mathsf{ak},\mathsf{sm},\mathsf{k},\mathsf{msg})) = 1\right] > \mathsf{negl}(\lambda) \quad ,$$

or the reliability of an undetectable algorithm substitution attack ASA against Π is at most negligible

$$\min_{\mathsf{ak},\mathsf{sm},\mathsf{k},\mathsf{msg}} \Pr\left[\mathsf{ASA.Ext}(\mathsf{ak}, \mathsf{ASA.Enc}^{\Pi}(\mathsf{ak},\mathsf{sm},\mathsf{k},\mathsf{msg})) \neq \mathsf{sm}\right] > 1 - \mathsf{negl}(\lambda) \quad .$$

Note that the definition of subversion-resilience corresponds to the security game in Figure 2.2, in which the watchdog Wa has a non-negligible advantage.
In the computational asymptotic setting of cryptography, we often rely on black-box reductions of adversaries to show that constructions must be secure. Optimally, we would want something similar for the subversion setting, showing that algorithms are subversion-resilient even if the implementation acts as a black box. Sadly, this is not possible. Russell et al. conjectured in [RTYZ16, RTYZ17] that black box defenses against algorithm substitution attacks may be impossible and Berndt and Liśkiewicz proved it in [BL17]. Any algorithm with sufficient entropy in its outputs is susceptible to steganographic attacks in a black-box setting. As ASAs model steganography over a certain type of channel, this means that ASAs always remain possible in a black-box scenario.
As such, the best case scenario we can achieve is a non-black-box-variant of subversion-resilience. The question then becomes how to define such a non-black-box framework. There have been multiple successful attempts in recent years, each with their own advantages and setbacks. Some examples include cryptographic reverse firewalls [MSD15], which act as a third party sanitizing messages via rerandomization, and self-guarding schemes [FM18], which split off a non-subverted initialization phase. However, in this thesis, we will focus on a third: the trusted amalgamation with split-program model.
First introduced in [RTYZ16] and refined in [RTYZ17], this type of model works by as-

suming that we can split provided implementations of a program into subprocedures, individually check these for subversions and then *glue* these securely together. As we will leverage this two-part model for security proofs, we provide a formal definition for both of them. We start with the split-program model.

Definition 2.11. Split-Program Model
Let Π be a randomized algorithm following specification $\widehat{\Pi}$. We assume now, that specification $\widehat{\Pi}$ can be described as a set of constantly many subprocedures $(\widehat{\pi}_1, \ldots, \widehat{\pi}_k)$.
When analyzing an implementation $\tilde{\Pi}$ of algorithm Π for subversions, this implementation must then be given by the adversary $\mathcal{A}$ as

$$\tilde{\Pi} := (\tilde{\pi}_1, \ldots, \tilde{\pi}_k) \quad ,$$

with the watchdog Wa being allowed to individually check each implementation.

Note that the split-program model only allows for a constant amount of subprocedures. This rules out that algorithms are broken down to a gate-by-gate level, meaning that subprocedures must be treated as simplified algorithms themselves. While a gate-by-gate analysis could result in stronger security guarantees, any arguments would necessarily rely on the security of the underlying hardware architecture. Additionally, working on a gate-by-gate level allows for easier concealment of malicious bootstrapping and control flow modifications. The result is, that an exponentially more complex notion of 'secure combination' is necessary in a gate-by-gate analysis than a subprocedure analysis. With the split program model we thus keep this last layer of abstraction intact.
Additionally, note that the split programming model does not inherently handle how to securely combine the subprocedures to a complete implementation. If we assume that the combination step is not secured, then it is possible to use a strategy similar to return-oriented programming [Sha07]. Each subprocedure on its own may be secure, but by combining them in unintended ways as gadgets, it may be possible to construct a subversion on the entire implementation.
The trusted amalgamation model addresses this exact issue by providing a secure amalgamation function Am, that is, by assumption, non-subverted. Using this amalgamation function to combine subprocedures then results in an implementation that follows the specifications control flows exactly. Similarly to the split-program model, we formally define the framework.

Definition 2.12. Trusted Amalgamation Model
Let Π be a randomized algorithm following specification $\widehat{\Pi} := (\widehat{\pi}_1, \ldots, \widehat{\pi}_k)$ and further let $\tilde{\Pi} := (\tilde{\pi}_1, \ldots, \tilde{\pi}_k)$ be an implementation of Π. Assume now that $\mathsf{Am}(\cdot)$ is a non-subverted, deterministic function called the amalgamation function.
Then, Am works by taking a set of subprocedures and amalgamating them corresponding to the algorithm's specification $\widehat{\Pi}$. The amalgamated implementation is given as

$$\mathsf{Am}(\tilde{\Pi}) := \mathsf{Am}(\mathsf{Am}(\tilde{\pi}_1), \ldots, \mathsf{Am}(\tilde{\pi}_k)) \quad ,$$

wherein each implementation $\tilde{\pi}_i$ may itself be an amalgamated implementation $\mathsf{Am}_i(\tilde{\pi}_i)$.

The usage of an amalgamation function in the trusted amalgamation model thus guarantees us that the combination of subprocedures does not introduce new subversions into the implementation. This means, that if all subprocedures of an implementation are subversion-resilient, then the amalgamation is subversion-resilient in the trusted amalgamation model.
For this model to work, we presume a trusted, non-subverted amalgamation function Am. However, it may not be obvious for such a function to exist. To prevent any subversions, we would at least want Am to contain as little complexity as possible. In the best case, Ams only task should be to guarantee the control flow.
Luckily, there exist two constructions fulfilling this condition. One of these constructions is described in [RTYZ17], while the other is described in [BCJ21]. Both amalgamation functions only handle inputs and outputs of subroutines, only requiring the use of a secure xor operation to do so. Whenever we use the trusted amalgamation model throughout this thesis, we will presume the use of one of these two constructions.
While these two constructions for amalgamation functions are of low complexity, they still rely on a trusted xor function. In general, most constructions will require the use of a small set of trusted, non-cryptographic operations. As an example, [BBD+23] require both trusted comparison and xor operations to achieve their results, arguing that both operations are simple enough to be realized in hardware in a trusted manner. Since we use a similar approach to [BBD+23] we will also assume both of these operations as trusted. Additionally, we will also require a trusted concatenation function to argue the security of our construction in Chapter 7. As a concatenation operation is of similar complexity to a comparison or xor, we analogously argue its security.
With all necessary notions for subversion-resilience handled, we will heavily rely on two more related statements throughout this thesis. For multiple steps of our constructions,

we will need to argue that a watchdog exists for deterministic algorithms operating on publicly known input distributions. While it seems obvious that a watchdog can simply sample the input distribution and then test the implementation of the deterministic algorithms on these inputs, a formal proof of this fact was given by Russel et al. in [RTYZ16]. For simplicity, we shall cite the lemma here for reference and refer to it when applicable later.

Lemma 2.13 ([RTYZ16]).
Let Π be a randomized algorithm following specification $\widehat{\Pi} := (\widehat{\pi}_1, \ldots, \widehat{\pi}_k)$. Consider an implementation $\tilde{\Pi} := (\tilde{\pi}_1, \ldots, \tilde{\pi}_k)$ of $\widehat{\Pi}$, where $\pi_1, \ldots, \pi_k$ are deterministic algorithms working on randomized inputs. Let then, for each security parameter λ, $X_\lambda^1, \ldots, X_\lambda^k$ be defined as their respective public input distribution.
Now, if there exists a $j \in [k]$ such that $\Pr[\tilde{\pi}_j(x) \neq \widehat{\pi}_j(x) \ : \ x \leftarrow\$ X_\lambda^j] = \delta$, then this can be detected by a PPT offline watchdog with a probability of at least δ.

The second statement is a theorem first proposed by Andrew Yao in [Yao77]. The theorem is colloquially known as Yao's Principle and relates the performance of randomized algorithms to deterministic algorithms. In general terms, it states that we can extract the randomness from a probabilistic algorithm and provide it via the input. We provide a simplified version of the theorem here, adapted from the modern write-ups on the theorem in [AB09, Wig19].

Theorem 2.14 (Yao's Principle (Simplified)).
When trying to find the optimal performance of a probabilistic protocol, rather than focusing on the best probabilistic algorithm for a worst-case input, we arrive at the same result when focusing on the best deterministic algorithm for an average-case input.

Yao's Principle may seem unrelated at first sight, but we can use it to great effect when working in the trusted amalgamation with split-program model. Explicitly, it guarantees us that we can consider the randomness generation to be a separate subprocedure, meaning that we can model all other subprocedures as deterministic algorithms working on randomized inputs. Russel et al. additionally showed in [RTYZ16] that such a split randomness generator can be constructed in the split-program model, even under subversion. This allows us to assume that, whenever we work in the trusted amalgamation with split-program model, we can access a non-subverted source of *good* randomness.
We continue with definitions and notations for obfuscation schemes.

2.3 Obfuscation

Next to steganography and algorithm substitution attacks, we will use obfuscation methods at multiple steps of our construction, namely throughout Chapters 4, 5 and 7. As such, we want to formally define all necessary notation and highlight some of the characteristics that apply when working with obfuscation schemes.

Obfuscation schemes, and program obfuscations in particular, were considered in parallel to other classic cryptographic problems such as encryption and authentication. While the other settings were able to achieve results in complexity-theory, such as the computationally asymptotic setting we are working within, there were no concrete results for the existence of obfuscation scheme for quite a long time [BGI+01].

The goal of obfuscation schemes is to produce some way to make any arbitrary program code *illegeable* to possible observers while keeping its functionality intact. Optimally, we would thus want this obfuscated program to act as a black-box to any observer, only retaining the input-output behavior of the original program while providing no actual information about its internal workings. We call such an obfuscation scheme a virtual black-box obfuscator [BGI+01]. A formal definition of a virtual black-box obfuscation can be found in Definition 2.15. Furthermore, the related security game formalizing the virtual black-box property can be found in Figure 2.3.

We note that obfuscation schemes are generally defined via circuit representations of programs to counteract the inherent informality of describing programs in natural language. In most works, circuits are denoted with a C [BGI+12, GR14, KMN+14], but as this notation clashes with definitions in the steganographic setting used within this thesis, most notably channels $\mathcal{C}$, we decided to use the notion of programs P and their implementations I. Unless explicitly stated otherwise, implementations I can be considered equivalent to the standard circuit definition.

The existence of virtual black-box obfuscation would allow for a wide variety of cryptographic constructions. Barak et al. describe in [BGI+01] usecases for watermarking, homomorphic encryption, random oracle models and translation from symmetric encryption schemes to public-key encryption. Sadly, however, virtual black-box obfuscation is impossible for a vast amount of programs. This impossibility result is, again, traceable to Barak et al. in [BGI+01], where the authors show that a virtual black-box obfuscation scheme can trivially be broken by two functions $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$ defined as

$$C_{\alpha,\beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0^\lambda & \text{else} \end{cases} \quad \text{and} \quad D_{\alpha,\beta}(C) := \begin{cases} 1 & \text{if } C(\alpha) = \beta \\ 0 & \text{else} \end{cases} .$$

Definition 2.15. Virtual Black-Box Obfuscation
Let $\lambda \in \mathbb{N}$ be the security parameter of the system and $\mathcal{P} = \{\mathcal{P}_\lambda\}$ the set of all polynomial-sized programs with parameterized input length $n(\lambda)$. Further, let $\mathcal{I} = \{\mathcal{I}_\lambda\}$ be a family of implementations over the subset of programs $\mathcal{I}_\lambda \subseteq \mathcal{P}$ and $\mathcal{D} = \{\mathcal{D}_\lambda\}$ a distribution ensemble over the implementations.
$\mathcal{O}$ then is a tuple of PPTMs $(\mathcal{O}.\mathsf{Gen}, \mathcal{O}.\mathsf{Obf})$ called a virtual black-box obfuscator if and only if for every $\lambda \in \mathbb{N}$ and every $I \in \mathcal{I}$ it satisfies the following statements:

- **Correctness**

  ϕ_1 : ***(perfect correctness)***
  For any Boolean circuit implementation $I \in \mathcal{I}$, the obfuscation given by $\mathcal{O}$ must behave identically on every possible input.

  $$\forall I \in \mathcal{I},\ \forall x \in \{0,1\}^{n(\lambda)} \ :\ \Pr_{\mathcal{O}}\left[\tilde{I}(x) = I(x) \mid \tilde{I} \leftarrow \mathcal{O}.\mathsf{Obf}(I)\right] = 1$$

  ϕ_2 : ***(functionality preserving)*** [BGI+12, BV16]
  For any Boolean circuit implementation $I \in \mathcal{I}$, the obfuscation given by $\mathcal{O}$ must behave identically over every possible input with overwhelming probability.

  $$\forall I \in \mathcal{I} \ :\ \Pr_{\mathcal{O}}\left[\forall x \in \{0,1\}^{n(\lambda)} : \tilde{I}(x) = I(x) \mid \tilde{I} \leftarrow \mathcal{O}.\mathsf{Obf}(I)\right] > 1 - \mathsf{negl}(\lambda)$$

  ϕ_3 : ***(weak functionality preserving)*** [KMN+14]
  For any Boolean circuit implementation $I \in \mathcal{I}$, the obfuscation given by $\mathcal{O}$ must behave identically on each possible input with overwhelming probability.

  $$\forall I \in \mathcal{I},\ \forall x \in \{0,1\}^{n(\lambda)} \ :\ \Pr_{\mathcal{O}}\left[\tilde{I}(x) = I(x) \mid \tilde{I} \leftarrow \mathcal{O}.\mathsf{Obf}(I)\right] > 1 - \mathsf{negl}(\lambda)$$

- **Polynomial Slowdown**
  There exists a polynomial q such that the running time of $\tilde{I} = \mathcal{O}.\mathsf{Obf}(I)$ is bounded by $q(|I|)$, wherein $|I|$ denotes the size of the implementation.

- **Virtual Black-Box**
  For every *(non-uniform)* polynomial size adversary $\mathcal{A}$, there exists a *(non-uniform)* polynomial size simulator $\mathcal{S}$ with oracle access to I such that for every distribution $D \in \mathcal{D}$:

  $$\left|\Pr_{I \leftarrow D, \mathcal{O}, \mathcal{A}}\left[\mathcal{A}(\mathcal{O}.\mathsf{Obf}(I)) = 1\right] - \Pr_{I \leftarrow D, \mathcal{S}}\left[\mathcal{S}^I(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

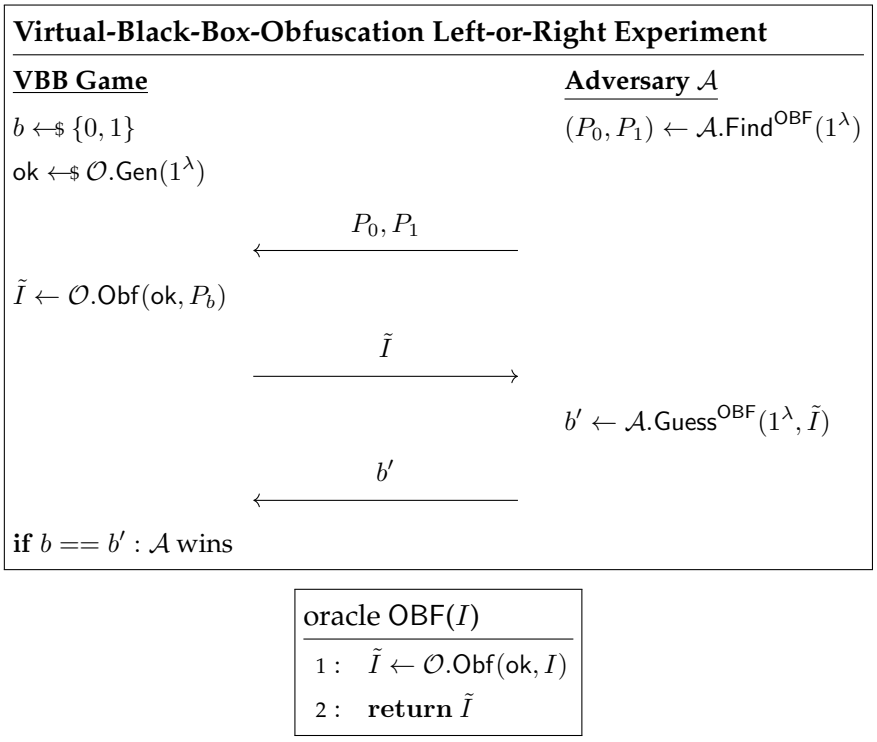


Figure 2.3: Depiction of the Virtual-Black-Box-Obfuscation Left-or-Right Experiment, abbreviated to VBB-Obf-LoR. An adversary $\mathcal{A}$ on the obfuscation scheme $\mathcal{O}$ chooses two implementations while given oracle access to the obfuscation method and queries $\mathcal{O}$ on them. The vbb game flips a random coin and, depending on its output, either obfuscates the left or right program and returns the result to $\mathcal{A}$. Adversary $\mathcal{A}$ must then distinguish which of their two programs was obfuscated. They win if they guess correctly.

However, this proof breaks down for some restricted classes of functions. As an example, [LPS04] and [BBC+14] showed that some types of point and evasive functions can be obfuscated. Point functions are functions that evaluate at exactly one input value to 1 and otherwise to 0, whereas evasive functions only return 1 with negligible probability on any input. Further examples can be found in [HMLS07] and [HRSV11], which explicitly focus on functions with use in cryptographic settings.
In conclusion, while arbitrary virtual black-box obfuscation is impossible, obfuscation in itself is not something to completely disregard. We will discuss one of its most notorious examples in the following section, indistinguishability obfuscation.

2.3.1 Indistinguishability Obfuscation

Indistinguishability obfuscation ($i\mathcal{O}$) describes a particular edge case in obfuscation schemes. Instead of obfuscating a family of multiple functions, we now consider only one singular function and the family of its implementations. Surprisingly, the impossibility proof for virtual black-box obfuscation from Barak et al. in [BGI+01] does not rule out creating an obfuscation scheme on this family of implementations. One can intuit this fact by considering the following trivial construction:
For a given program P, list all possible implementations I in ascending order of size. When calling $i\mathcal{O}$ on any of these implementations I, simply return whichever implementation occupies the first position in the listing as its obfuscation. Evidently, the output of $i\mathcal{O}$ then perfectly hides which implementation was used as its input while keeping the functionality of the program intact.
Indistinguishability obfuscation is formally defined in the following.

Definition 2.16. Indistinguishability Obfuscation ($i\mathcal{O}$)
Let $\lambda \in \mathbb{N}$ be the security parameter of the system and $\mathcal{P} = \{\mathcal{P}_\lambda\}$ the set of all polynomial-sized programs with parameterized input length $n(\lambda)$. In addition, let $\mathcal{I} = \{P\}$ be a family of implementations of the same program $P \in \mathcal{P}$ and $\mathcal{D} = \{\mathcal{D}_\lambda\}$ a distribution ensemble over the implementations.
$i\mathcal{O}$ then is a tuple of PPTMs $(i\mathcal{O}.\mathsf{Gen}, i\mathcal{O}.\mathsf{Obf})$ called an indistinguishability obfuscator if and only if for every $\lambda \in \mathbb{N}$ and every $I \in \mathcal{I}$ it satisfies the following statements:

- **Correctness**

  (see the correctness property of Definition 2.15)

- **Polynomial Slowdown**

  (see the polynomial slowdown property of Definition 2.15)

- **Indistinguishability**
  For every same-size pair of Boolean circuit implementations of P, the two obfuscation distributions produced by $i\mathcal{O}$ on input of either of the two circuits must be computationally indistinguishable for any poly-time adversary $\mathcal{A}$.

$$\forall I_0, I_1 \in \mathcal{I},\ \forall \mathcal{A}\ :\ \left| \Pr_{i\mathcal{O}}\left[\mathcal{A}(i\mathcal{O}.\mathsf{Obf}(I_0)) = 1\right] - \Pr_{i\mathcal{O}}\left[\mathcal{A}(i\mathcal{O}.\mathsf{Obf}(I_1)) = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

The security game for an $i\mathcal{O}$ scheme is then defined via its indistinguishability property:

We can construct a typical left-or-right experiment to achieve this. The adversary $\mathcal{A}$ is allowed to choose any two different implementations I_0, I_1 of the program P to send to the game. At the same time, the game flips a coin. Upon receiving the two implementations, the game chooses which implementation to obfuscate via the prior coin flip. The obfuscated program is then sent back to the adversary, who needs to answer which of the two implementations was obfuscated. A visual representation of this game can be found in Figure 2.4.

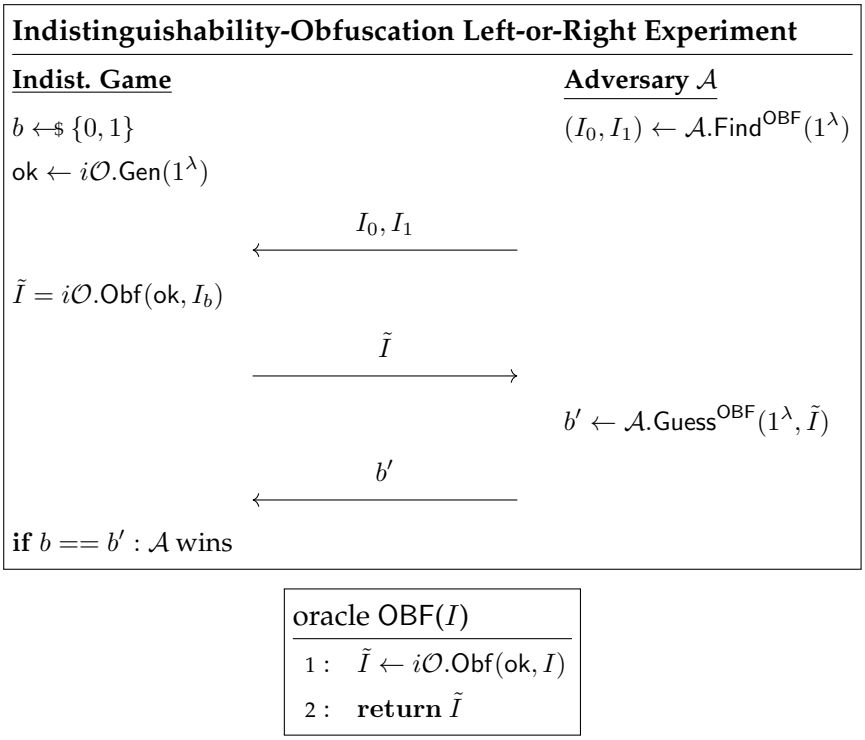


Figure 2.4: Depiction of the Indistinguishability-Obfuscation Left-or-Right Experiment, abbreviated to $i\mathcal{O}$-LoR. The game runs analogously to the VBB-Obf-LoR in Figure 2.3, however, the chosen implementations I_0 and I_1 may only be of the same program P defining the obfuscation scheme $i\mathcal{O}$. The adversary $\mathcal{A}$ still only wins if $b' = b$.

While the existence of indistinguishability obfuscation was known since the results of [BGI+01], for a couple years, no one knew how to effectively leverage $i\mathcal{O}$. The existence of $i\mathcal{O}$ was thus only considered an interesting but not utilizable exception to the impossibility results for virtual black-box obfuscation schemes [BGI+12, GR14].
However, $i\mathcal{O}$ turned out to be surprisingly versatile as Sahai and Waters were able to show in [SW14]. Generally, $i\mathcal{O}$ only works for implementations of some fixed program P. However, if we now construct an implementation that only implements P with over-

whelming probability, we can still obfuscate this modified program with almost certainty. This allows us, contrary to intuition, to leverage indistinguishability obfuscation as an intermediary step in game hopping proofs.
Research since the original paper from Sahai and Waters has shown that most cryptographic primitives can be constructed from $i\mathcal{O}$. We refer to the works in [JLS21] and [GJLS21] for more comprehensive lists of constructions.
Besides indistinguishability obfuscation, we will use two other types of obfuscation throughout this thesis, namely malicious and auditable obfuscation. They will be the focus of the next section.

2.3.2 Malicious & Auditable Obfuscation

Malicious and auditable obfuscation, $\mathcal{MO}$ and $\mathcal{AO}$ respectively, model a similar concept to algorithm substitution attacks and subversion-resilience on obfuscation schemes. Obfuscation, even though not cryptographically secure, is often used in real applications, most commonly to prevent the reverse engineering of software. Due to the complexity of obfuscation schemes, it is common practice to not implement individual obfuscation schemes, but rather rely on third party tools.
However, in general, we can not assume any third party to be trustworthy. A bad actor could program their obfuscator in such a way that any inserted program is modified in some unintended way. This modification can range from small errors in some outputs up to the embedding of malware that is executed alongside the original program [BG23]. We call an obfuscation with such modifications a malicious obfuscation.
Malicious obfuscations have been a part of literature for a while, being referenced in works such as [BGJS16]. However, $\mathcal{MO}$ often is only described in natural language and not formally defined. A first formal definition for malicious obfuscation was given in [BG23] by Banerjee and Galbraith. They build their definition for $\mathcal{MO}$ from a perspective of correctness, stating that malicious obfuscation is foremost a violation of the correctness guarantees of the underlying obfuscation scheme. We adopt this formal definition in Definition 2.17.
The security of a malicious obfuscation scheme is defined as the inability of a distinguisher to differentiate between a malicious and honest obfuscation. We model the modifications made during the malicious obfuscation as a type of embedding done with some auxiliary information aux. The strongest security setting for a malicious obfuscation scheme then allows the distinguisher to specify what this embedding should entail.
A visual representation of this chosen-embedding game can be found in Figure 2.5. We denote the distinguisher as an auditor Au on the obfuscation scheme. The auditor is given an embedding oracle and is allowed to provide the embedding used in the challenge via sm.

Definition 2.17. Malicious Obfuscation ($\mathcal{MO}$)
Let $\lambda \in \mathbb{N}$ be the security parameter of the system and $\mathcal{P} = \{\mathcal{P}_\lambda\}$ the set of all polynomial-sized programs with parameterized input length $n(\lambda)$. Further, let $\mathcal{O}$ be an obfuscator satisfying Definition 2.15 with correctness ϕ_i on the family of implementations $\mathcal{I} = \{\mathcal{I}_\lambda\}$ with $\mathcal{I}_\lambda \subseteq \mathcal{P}$ and distribution ensemble $\mathcal{D} = \{\mathcal{D}_\lambda\}$ over the implementations.
$\mathcal{MO}$ then is a tuple of PPTMs ($\mathcal{MO}.\mathsf{Gen}, \mathcal{MO}.\mathsf{Emb}, \mathcal{MO}.\mathsf{Trig}$) called a malicious obfuscator for the family $\mathcal{I}$ and distribution $\mathcal{D}$ if and only if on auxiliary input $\mathsf{aux} \in \{0,1\}^\lambda$ it satisfies the following statements:

- **Correctness violation**
  For any choice of aux, $\mathcal{MO}.\mathsf{Emb}(1^\lambda, \mathsf{aux}, \cdot, \mathsf{ok})$ does not satisfy ϕ_i.
- **Indistinguishability**
  For every probabilistic polytime distinguisher, also called auditor Au, the output from $\mathcal{MO}.\mathsf{Emb}$ and $\mathcal{O}.\mathsf{Obf}$ must be computationally indistinguishable.

$$\left| \Pr_{\mathsf{aux},\mathsf{Au},\mathcal{MO}} \left[\mathsf{Au}^{\mathcal{MO}.\mathsf{Emb}(1^\lambda,\mathsf{aux},\cdot,\mathsf{ok})} = 1 \right] - \Pr_{\mathsf{Au},\mathcal{O}} \left[\mathsf{Au}^{\mathcal{O}.\mathsf{Obf}(1^\lambda,\cdot,\mathsf{ok})} = 1 \right] \right| \leq \mathsf{negl}(\lambda)$$

We note the similarity of the game's structure to the security games of stegosystems StS and algorithm substitution attacks ASA. As mentioned before, malicious obfuscation can be intuited as an algorithm substitution attack on an obfuscation scheme $\mathcal{O}$. However, this similarity in their respective security games hints at an even deeper relation. Precisely, we will show that this intuition of similarity actually is an equivalence in Chapter 4.
With the problem of malicious obfuscation raised and its intuition as an algorithm substitution attack on obfuscation schemes presented, the next step should be to try and find a modeling that can be intuited as subversion-resilience on obfuscation schemes. In literature, first attempts at this were classified under the notion of verifiable obfuscation, such as the model in [CV09]. However, these definitions do not fully protect against our definition of malicious obfuscation as a violation of correctness in their entirety.
Banerjee and Galbraith also realized this and proposed auditable obfuscation as a solution in [BG23]. Analogously to watchdogs in the ASA setting, they introduce an auditor that looks at all outputs of an obfuscation scheme $\mathcal{O}$ and probes it for malicious embeddings. Note that, similar to watchdogs on algorithm substitution attacks, this auditor also is unable to completely mitigate malicious obfuscations, only providing an upper bound to their effectiveness in most cases.
The auditor works by rejecting any obfuscation containing malicious embeddings, which,

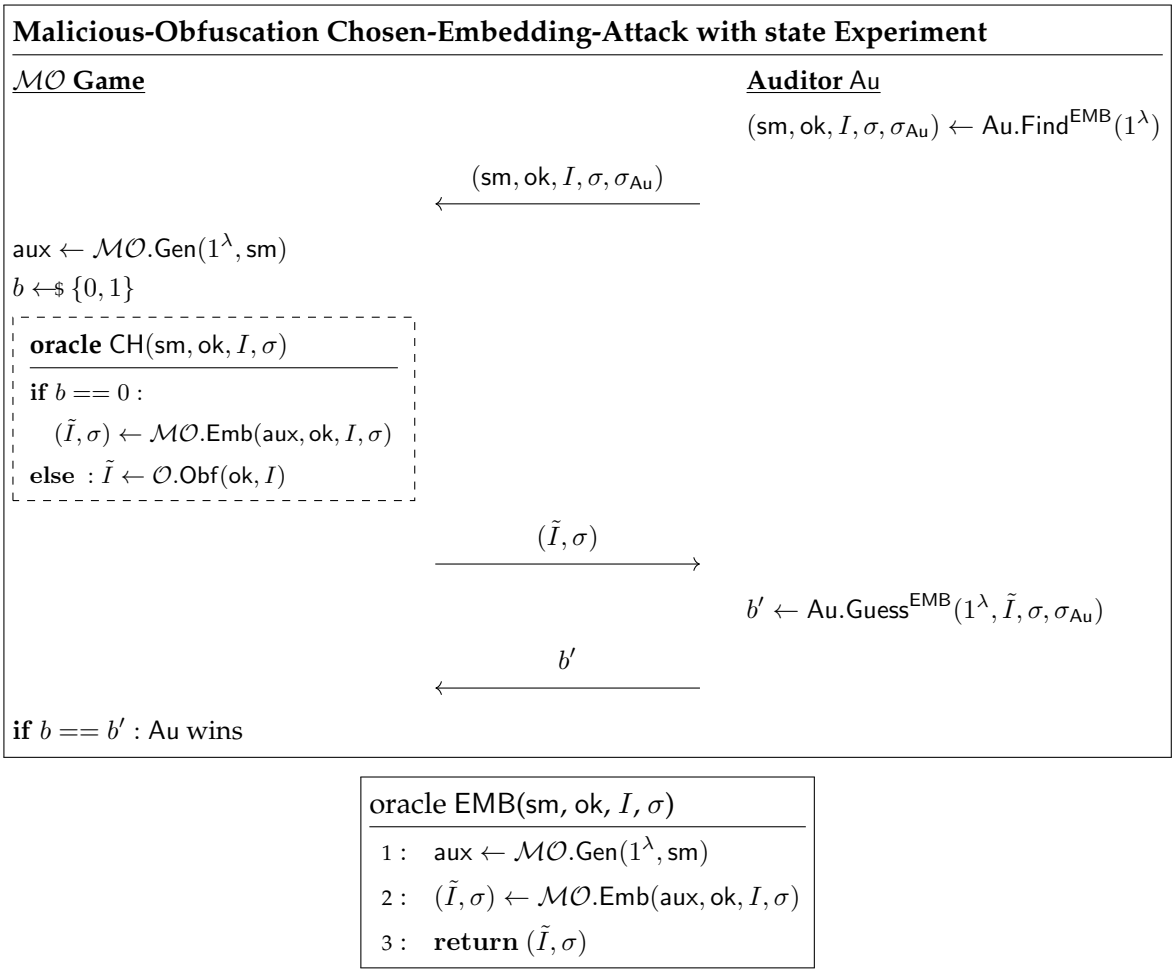


Figure 2.5: Depiction of the Malicious-Obfuscation Chosen-Embedding-Attack with state Experiment, abbreviated to $\mathcal{MO}$-CEA-σ. An auditor Au tries to determine wether the output of an obfuscator $\mathcal{O}$ is malicious or not. For this, they are given an embedding oracle providing them with malicious obfuscations. Au uses them to create a *best case* scenario and queries their challenge oracle CH accordingly. Depending on the output of CH, the auditor must then decide and wins if they guess correctly.

as previously defined, modify the obfuscation's correctness class while keeping its indistinguishability property intact. Only if no malicious embedding is found, meaning any malicious obfuscation scheme on the obfuscator was unable to effectively modify the correctness, does the auditor accept an obfuscation. An auditor-accepting obfuscation scheme is thus described by the provable absence of any functional malicious obfuscation scheme against it. We call such obfuscation schemes *auditable* or *auditable obfuscation* schemes [BG23] and define them as follows:

Definition 2.18. Auditable Obfuscation ($\mathcal{AO}$)
Let $\lambda \in \mathbb{N}$ be the security parameter of the system and $\mathcal{P} = \{\mathcal{P}_\lambda\}$ the set of all polynomial-sized programs with parameterized input length $n(\lambda)$. Further, let $\mathcal{I} = \{\mathcal{I}_\lambda\}$ be a family of implementations over the subset of programs $\mathcal{I}_\lambda \subseteq \mathcal{P}$ and $\mathcal{D} = \{\mathcal{D}_\lambda\}$ a distribution ensemble over the implementations.
$\mathcal{AO}$ then is a tuple of PPTMs $(\mathcal{AO}.\mathsf{Gen}, \mathcal{AO}.\mathsf{Obf}, \mathcal{AO}.\mathsf{Verify})$ called an auditable obfuscator if and only if for every $\lambda \in \mathbb{N}$ and every $I \in \mathcal{I}$ it satisfies the following statements:

- **Correctness**

  (see the correctness property of Definition 2.15)

- **Polynomial Slowdown**

  (see the polynomial slowdown property of Definition 2.15)

- **Virtual Black-Box**

  (see the virtual black-box property of Definition 2.15)

- **Verifiability**
  For every $(\tilde{I}, \nu) \leftarrow \mathcal{AO}.\mathsf{Obf}(1^\lambda, I)$, $\mathcal{AO}.\mathsf{Verify}(1^\lambda, I, \tilde{I}, \nu) \rightarrow \{0, 1\}$.

- **Soundness**
  For every *(non-uniform)* polynomial size adversary $\mathcal{A}$, if

  $$\left[(\tilde{I}', \nu) \leftarrow \mathcal{A}(1^\lambda, I)\right] \wedge \left[1 \leftarrow \mathcal{AO}.\mathsf{Verify}(1^\lambda, I, \tilde{I}', \nu)\right]$$

  then $\tilde{I}'$ satisfies ϕ_i.

Note that we switch the virtual black-box property of $\mathcal{AO}$ with the following indistinguishability property.

- **VBB Indistinguishability**
  For any two implementations I_0, I_1, for every *(non-uniform)* polynomial size adversary $\mathcal{A}$, there exists a *(non-uniform)* polynomial size simulator $\mathcal{S}$ with either oracle access to I_0 or I_1, such that the term

  $$\left| |\Pr_{\mathcal{AO}}[\mathcal{A}(\mathcal{AO}.\mathsf{Obf}(I_0)) = 1] - \Pr_{\mathcal{AO}}[\mathcal{A}(\mathcal{AO}.\mathsf{Obf}(I_1)) = 1]| - |\Pr_{\mathcal{S}}[\mathcal{S}^{I_0}(1^\lambda) = 1] - \Pr_{\mathcal{S}}[\mathcal{S}^{I_1}(1^\lambda) = 1]| \right|$$

  is upper bounded by a negligible term.

In Chapter 5, we will show that an indistinguishability obfuscation scheme can be constructed from auditable obfuscation. However, compared to $\mathcal{AO}$ and virtual black-box obfuscation, $i\mathcal{O}$ uses an indistinguishability property instead of a virtual black-box property. The switch from the standard virtual black-box property to its vbb indistinguishability property thereby maps better between the two settings.

Auditable-Obfuscation Adaptive-LoR-or-Forgeability-Attack Experiment

$\mathcal{AO}$ **Game**		**Adversary** $\mathcal{A}$
$I_c \leftarrow \mathsf{Au}_{\mathcal{AO}}(1^\lambda)$		
	$\xrightarrow{I_c}$	
$b \leftarrow\$ \{0,1\}$		$(I_0, I_1) \leftarrow \mathcal{A}.\mathsf{Find}^{\mathsf{OBF},\mathsf{VER}}(1^\lambda)$
$\mathsf{ok} \leftarrow \mathcal{AO}.\mathsf{Gen}(1^\lambda)$		
	$\xleftarrow{(I_0, I_1)}$	
if $I_0 == I_c$ **or** $I_1 == I_c$:		
abort		
else : $(\tilde{I}, \nu) \leftarrow \mathcal{AO}.\mathsf{Obf}(\mathsf{ok}, I_b)$		
	$\xrightarrow{(\tilde{I}, \nu)}$	
		$(b', (\tilde{I_c}', \nu')) \leftarrow \mathcal{A}.\mathsf{Guess}^{\mathsf{OBF},\mathsf{VER}}(1^\lambda, I_c, \tilde{I}, \nu)$
	$\xleftarrow{(b', (\tilde{I_c}', \nu'))}$	
if $\mathcal{AO}.\mathsf{Verify}(1^\lambda, I_c, \tilde{I_c}', \nu') == 1$		
or $b == b'$: $\mathcal{A}$ wins		

oracle $\mathsf{OBF}(I)$	oracle $\mathsf{VER}(I, \tilde{I}, \nu)$
1 : $(\tilde{I}, \nu) \leftarrow \mathcal{AO}.\mathsf{Obf}(\mathsf{ok}, I)$	1 : $b \leftarrow \mathcal{AO}.\mathsf{Verify}(1^\lambda, I, \tilde{I}, \nu)$
2 : **return** $(\tilde{I}, \nu)$	2 : **return** b

Figure 2.6: Depiction of the Auditable-Obfuscation Adaptive-LoR-or-Forgeability-Attack Experiment, abbreviated to $\mathcal{AO}$-Adap-LRFA. An auditor Au chooses a challenge element from the set $\mathcal{I}$ and sends it to an adversary $\mathcal{A}$. That adversary may then choose a pair of two different implementations as the elements of a left-or-right game and query a secondary challenge from $\mathcal{AO}$ on them. $\mathcal{AO}$ auditably obfuscates one element of the secondary challenge and returns the output to $\mathcal{A}$. Adversary $\mathcal{A}$ can now both forge an obfuscation for the first challenge and try and distinguish the second challenge. If $\mathcal{A}$ wins either of those, they win the entire game.

The vbb indistinguishability can be traced back to Barak et al. in [BGI+01], where they provide it besides the standard virtual black-box definition. However, while Barak et al. note that the property appears to result in a slightly weaker notion of obfuscation, they do not elaborate further on it. As we will show that even this, at worst, weaker notion is sufficient for our purposes, we can directly infer that the stronger setting would also suffice.
With this, we can handle the security game for auditable obfuscation. It is important to note that we need to play two games at the same time, one for the vbb indistinguishability and one for the soundness. We can combine these two games into an adaptive setting, in which we give an adversary a challenge in both settings, only one of which they need to succeed at to win the game. A visual representation of the game can be found in Figure 2.6.
This concludes the introduction of all necessary obfuscation preliminaries. We continue with the last few definitions and notations regarding pseudorandomness and encryption.

2.4 Pseudorandomness & Encryption

The last two preliminary notions we need to introduce are the concept of pseudorandomness and encryption schemes. We group these two concepts as they are both inherently linked with the notion of randomness in cryptography. In most of the field, and especially in encryption schemes, randomness is a necessity. For example, deterministic (*stateless*) encryption schemes $\mathcal{E}$ can only provide provable security guarantees if they are executed on one message exactly once [KL14]. For any encryption scheme $\mathcal{E}$ that we want to use multiple times some method of making the encryption probabilistic is needed.
While theoretical settings allow us to presume a source of good randomness, real applications leave us unable to produce *true* random samples. As such, we need to find something that is as close to the real thing as possible. This is where the notion of (*cryptographic*) pseudorandomness comes into play. Note that, unless otherwise stated, we will adapt definitions and security games throughout this section from [KL14].

2.4.1 Pseudorandomness

Pseudorandomness is generally defined over a series of efficient outputs that should "look" random. More formally defined, we want (*cryptographic*) pseudorandomness to be deterministically computable in polytime, but result in an output that is computationally indistinguishable from true randomness. While non-cryptographic pseudorandomness has been researched for decades, most constructions do not fulfill the strict requirements for a cryptographic setting.
The first constructions usable in cryptographic proofs only came to be in the early 1980s

through the work of Blum and Micali in [BM84] and Yao in [Yao82]. Both constructions leverage a short true random seed s as an input to a generating function G. Based on s, the generator G then produces a long output of pseudorandom bits. We fittingly call such a generator a pseudorandom (*number*) generator PRG.

Definition 2.19. Pseudorandom Generator (PRG)
Let λ be the security parameter. G then is a deterministic PPTM with inputs of size $n(\lambda)$, output size $m >> n(\lambda)$ called a Pseudorandom Generator if and only if for every $\lambda \in \mathbb{N}$ it satisfies the following statements:

- **Computability**
  The output of G is, knowing some input s, efficiently computable by a deterministic algorithm.
- **Indistinguishability**
  The advantage of any PPT adversary $\mathcal{A}$ to distinguish the output of G from a uniform random distribution over $\{0,1\}^m$ is only negligible, meaning:

$$\left|\Pr_s[\mathcal{A}(G(s)) = 1] - \Pr_r[\mathcal{A}(r) = 1]\right| \leq \mathsf{negl}(\lambda)$$

The security of a PRG is then given by a security game over its indistinguishability property. The game first flips a coin on wether to use the PRG G or true randomness. Afterwards, the adversary is sent an output v, which either is the result of the PRG on a uniformly sampled seed s or uniformly randomly chosen. The setting is thereby dependent on the result of the coin flip. Lastly, the adversary must answer whether the received output was produced by G or just a random string. A visualization of the security game for a PRG is provided in Figure 2.7.
While PRGs are a versatile primitive in cryptographic instructions, we can not describe every use case with them. Pseudorandom generators allow us to produce random-like output strings, but in some cases, we need a more generalized notion of a pseudorandom function F. Such a pseudorandom function should be indistinguishable from a real random function f, that, on each input x, uniformly samples some output y.
PRFs for use in cryptography were first formalized in [GGM86], where a PRF F is defined as a keyed function, evaluated on inputs x. The key k then acts as the source of randomness, similarly to the seed s for PRGs.

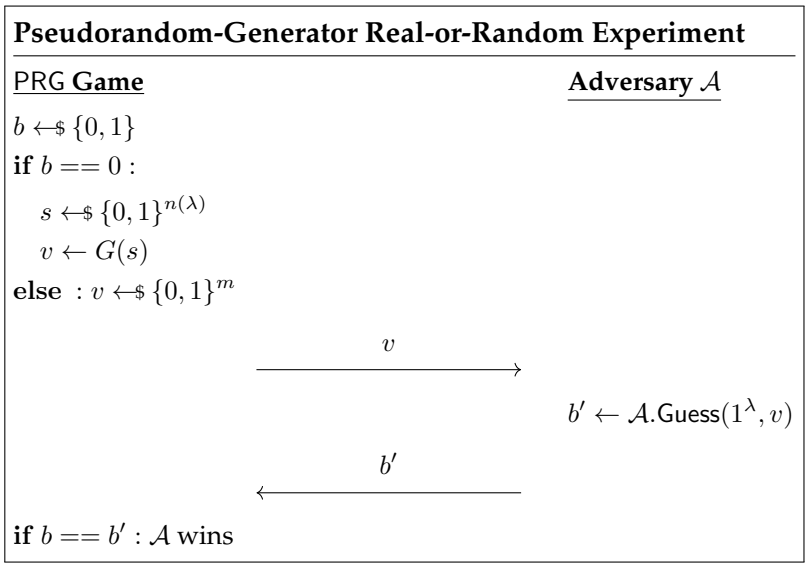


Figure 2.7: Depiction of the Pseudorandom-Generator Real-or-Random Experiment, abbreviated to PRG-RoR. The game flips a coin and, depending on the result, either samples a seed s and evaluates the PRG at s or samples a random string of equal length. The output is sent to an adversary $\mathcal{A}$. Their goal is to distinguish whether the output was produced by a PRG or not. If $\mathcal{A}$ guesses correctly, they win the game.

We formally define such a keyed PRF as follows:

Definition 2.20. Pseudorandom Function (PRF)
Let λ be the security parameter and $n(\lambda)$ some length. F then is a deterministic PPTM, taking a key $\mathsf{k} \in \{0,1\}^{n(\lambda)}$ as well as some value $x \in \{0,1\}^{n(\lambda)}$ as inputs to produce an output $y \in \{0,1\}^{n(\lambda)}$. F is called a pseudorandom function if and only if for every $\lambda \in \mathbb{N}$ it satisfies the following:

- **Computability**
  The output of F is, knowing the inputs k and x, efficiently computable by a deterministic algorithm.
- **Indistinguishability**
  The advantage of any PPT adversary $\mathcal{A}$ to distinguish the output of $F(\mathsf{k}, \cdot)$ on any input x from a uniform random distribution over $\{0,1\}^m$ is only negligible, meaning:
  $$\forall x \in \{0,1\}^{n(\lambda)} \ : \ \left| \Pr_{\mathsf{k}}[\mathcal{A}(F(\mathsf{k}, x)) = 1] - \Pr_{r}[\mathcal{A}(r) = 1] \right| \leq \mathsf{negl}(\lambda)$$

The security of the PRF is, again, given via its indistinguishability property. However, as mentioned before, in this game, we do not distinguish a PRF from a random string but rather a random function f. In further difference to the PRG game, we provide the adversary $\mathcal{A}$ with an evaluation oracle, with which $\mathcal{A}$ can query the PRF F in a black-box manner. A visualization for the security game of a PRF is provided in Figure 2.8.

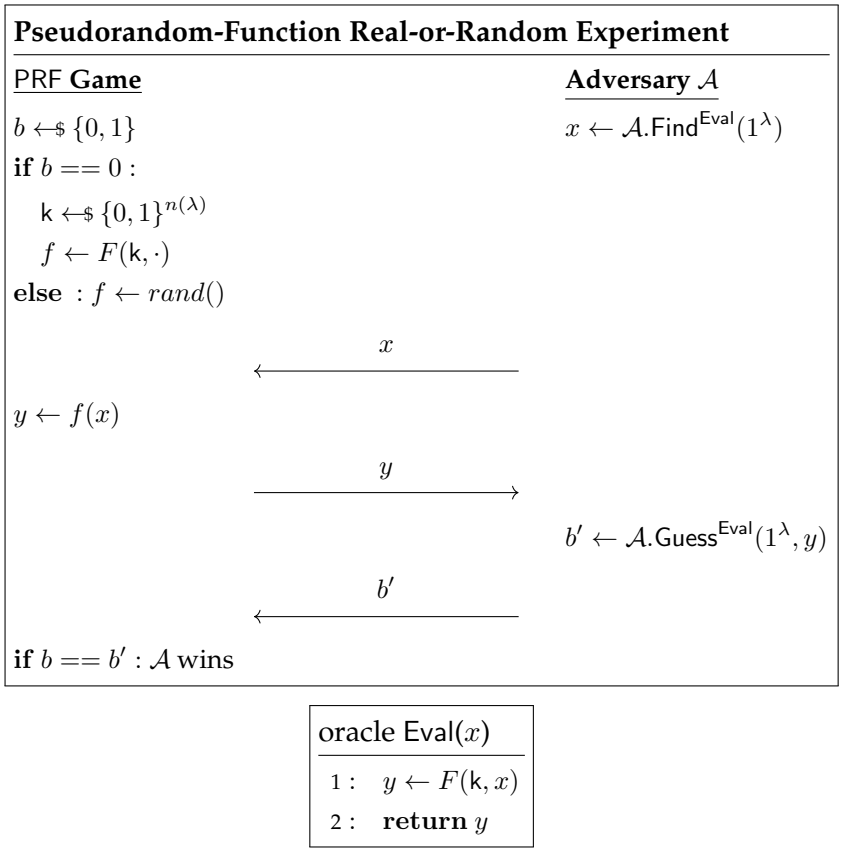


Figure 2.8: Depiction of the Pseudorandom-Function Real-or-Random Experiment, abbreviated to PRF-RoR. An adversary $\mathcal{A}$ tries to distinguish whether the output of the game is real, being the output of a pseudorandom function F, or truly random. For this, they are allowed to query the game on a chosen input x. To find x, $\mathcal{A}$ is allowed to query an evaluation oracle that provides them outputs of F. The game either evaluates F on x or samples a random value depending on a uniformly chosen bit and returns the output y afterwards. $\mathcal{A}$ must then guess whether y is real or random. If the adversary guesses correctly, they win the game.

In most of cryptography, when discussing pseudorandomness, we will either use a PRG, a PRF or a combination of the two primitives. However, some constructions require another slightly different notion of pseudorandomness. For some use case, it may be of benefit to

allow a third party to execute most of a PRF but hide some secret positions. Informally, we want to black out the results for some of the possible input values, but allow the rest to be evaluated correctly.
The idea behind such a PRF is referred to as a puncturable pseudorandom function, abbreviated to PPRF. These functions were simultaneously proposed by [BGI14, BW13] and [KPTZ13] as a type of constrained PRF that works by modifying the key of a PRF in such a way that only parts of the PRF can be evaluated on them. Such punctured keys are denoted with $\mathsf{k}\{x\}$, where x corresponds to the punctured input value. Note that, for simplicity, we write $\mathsf{k}\{\mathsf{punc}\}$ if multiple values are punctured.
Similarly to PRGs and PRFs, we provide a formal definition for PPRFs:

Definition 2.21. Punctured Pseudorandom Function (PPRF)
Let λ be the security parameter and $n(\lambda)$ some length. F then is a deterministic PPTM, taking a key $\mathsf{k} \in \{0,1\}^{n(\lambda)}$ as well as some value $x \in \{0,1\}^{n(\lambda)}$ as inputs to produce an output $y \in \{0,1\}^{n(\lambda)}$. F is called a puncturable pseudorandom function if and only if for every $\lambda \in \mathbb{N}$ it satisfies the following:

- **Computability**
  The output of F is, knowing the inputs k and x, efficiently computable by a deterministic algorithm.
- **Indistinguishability**
  The advantage of any PPT adversary $\mathcal{A}$ to distinguish the output of $F(\mathsf{k}, \cdot)$ on any input x from a uniform random distribution over $\{0,1\}^m$ is only negligible, meaning:
  $$\forall x \in \{0,1\}^{n(\lambda)} \ : \ \left| \Pr_{\mathsf{k}}[\mathcal{A}(F(\mathsf{k}, x)) = 1] - \Pr_{r}[\mathcal{A}(r) = 1] \right| \leq \mathsf{negl}(\lambda)$$
- **Puncturable Indistinguishability**
  The advantage of any PPT adversary $\mathcal{A}$ with access to punctured key $\mathsf{k}\{\mathsf{punc}\}$, where punc describes a set of puncturings, to distinguish the output of F at a punctured position from a uniform random distribution over $\{0,1\}^m$ is only negligible, meaning:
  $$\forall x \in \mathsf{punc} \ : \ \left| \Pr_{\mathsf{k}}[\mathcal{A}(\mathsf{k}\{\mathsf{punc}\}, F(\mathsf{k}, x)) = 1] - \Pr_{r}[\mathcal{A}(\mathsf{k}\{\mathsf{punc}\}, r) = 1] \right| \leq \mathsf{negl}(\lambda)$$

We can now define the security game for punctured PRFs. This time, however, we define

the security of the function over its puncturings rather than its indistinguishability. An adversary $\mathcal{A}$ on a PPRF F should not be able to reconstruct a punctured output $y = F(k, x)$, even if they know F and a punctured key $\mathsf{k}\{x\}$. The security game then works by allowing the adversary to specify a challenge position to puncture. The PPRF is deemed secure if, even in this scenario, the adversary's advantage is negligible. A visualization for the security game of a PPRF is provided in Figure 2.9.

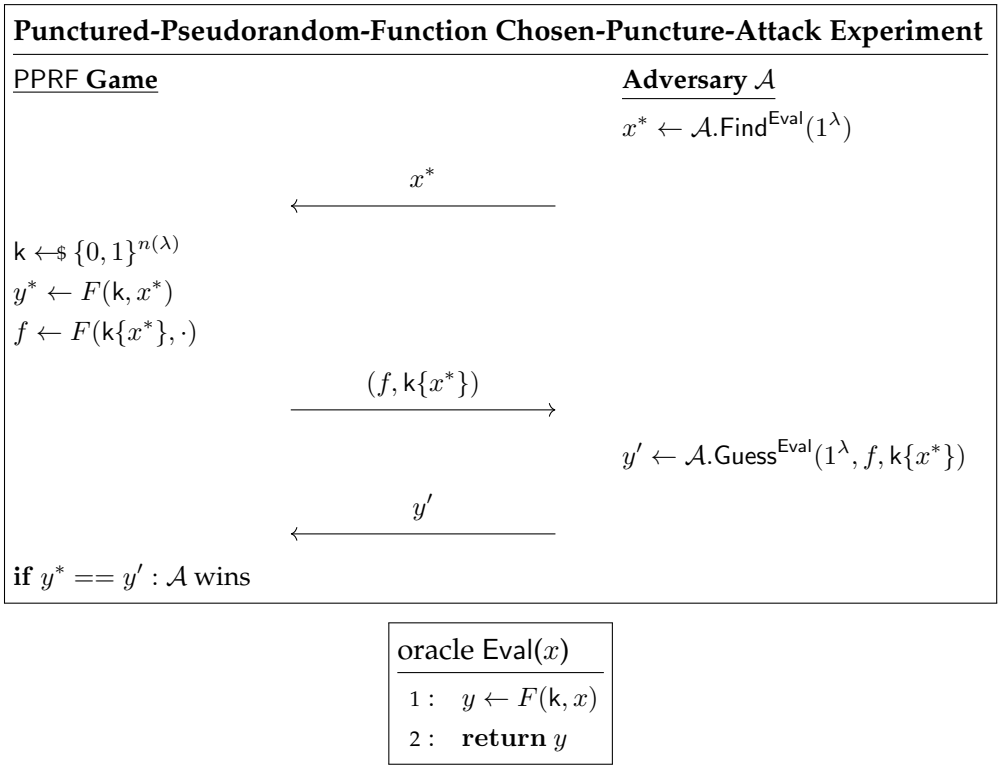


Figure 2.9: Depiction of the Punctured-Pseudorandom-Function Chosen-Puncture-Attack Experiment, abbreviated to PPRF-CPA. An adversary $\mathcal{A}$ tries to forge a PRF on a punctured point. For this, $\mathcal{A}$ is allowed to choose the puncturing position x^* to their liking. The game, after receiving position x^*, evaluates PRF on it and generates the punctured key $\mathsf{k}\{x^*\}$. Afterwards, the game provides $\mathcal{A}$ with the PRF and $\mathsf{k}\{x^*\}$. $\mathcal{A}$ now needs to force the punctured output. If they do so correctly, they win the game.

While it may at first not be intuitive whether such types of pseudorandom functions exist, it has been shown that the GGM construction for PRFs [GGM86] can easily be adapted to this punctured setting [BGI14, BW13, KPTZ13].
With this, we have handled all necessary notions of pseudorandomness used throughout this thesis. We continue with the definition of encryption schemes and their security.

2.4.2 Encryption Schemes

The encryption scheme model is one of the most commonly known cryptographic primitives, the secure communication between two parties. The encrypted messages should be protected from unauthorized access and modifications. When talking about encryption schemes, we generally distinguish between two types: private-key symmetric encryption and public-key asymmetric encryption. Private-key symmetric encryption, as the name implies, uses a shared private key k to encrypt and decrypt the traffic between the two parties. Public-key asymmetric encryption on the other hand relies on a key pair $(\mathsf{pk}, \mathsf{sk})$ for each party, consisting of a public key pk and secret key sk.
While we do indirectly use symmetric encryption throughout this thesis, we do not focus on it. As such, we omit a formal definition and refer the reader to the writeup on symmetric encryption in [KL14]. Inversely, one of our main results is the construction of a subversion-resilient IND-CCA-secure public-key encryption scheme in Chapter 7. As such, we provide a formal definition of it.

Definition 2.22. Public Key Encryption Scheme (PKES/$\mathcal{E}$)
Let $\lambda \in \mathbb{N}$ be a security parameter. A public-key encryption scheme $\mathcal{E}$ then is a triple of PPTMs $(\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$, wherein

- $\mathcal{E}.\mathsf{Gen}$ is a *key-generation* algorithm taking the security parameter to produce a pair of public and secret keys.
$$\mathcal{E}.\mathsf{Gen}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$$
- $\mathcal{E}.\mathsf{Enc}$ is an *encryption* algorithm that takes a public key pk and an encryption message $\mathsf{msg} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ to produce a ciphertext $c \in \{0,1\}^{\mathsf{poly}(\lambda)}$.
$$\mathcal{E}.\mathsf{Enc}(\mathsf{pk}, \mathsf{msg}) \rightarrow c \in \{0,1\}^{\mathsf{poly}(\lambda)}$$
- $\mathcal{E}.\mathsf{Dec}$ is a *decryption* algorithm taking a secret key sk and ciphertext c to produce a recovered message msg'.
$$\mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c) \rightarrow \mathsf{msg}' \in \{0,1\}^{\mathsf{poly}(\lambda)}$$

The security of an encryption scheme, similarly to the other primitives we tackled so far, is defined via security games. However, encryption differs slightly due to not having one common notion of security. Depending on the assumed strength of the adversary, we

play different security games. The most common types of security notions are IND-CPA security and IND-CCA security. In the IND-CPA setting, we restrict the adversary by only providing them with an encryption oracle and no information about the decryption function. In the IND-CCA setting, we additionally provide them with a decryption oracle. Note that the IND-CCA setting implies a strictly stronger adversary than the IND-CPA setting. Similar to symmetric encryption schemes, while we do mention IND-CPA security multiple times throughout this thesis, we do not focus on it. We will thus also omit its game visualization and refer the reader to its writeup in [KL14].
We do provide a visualization of the IND-CCA security game in Figure 2.10.

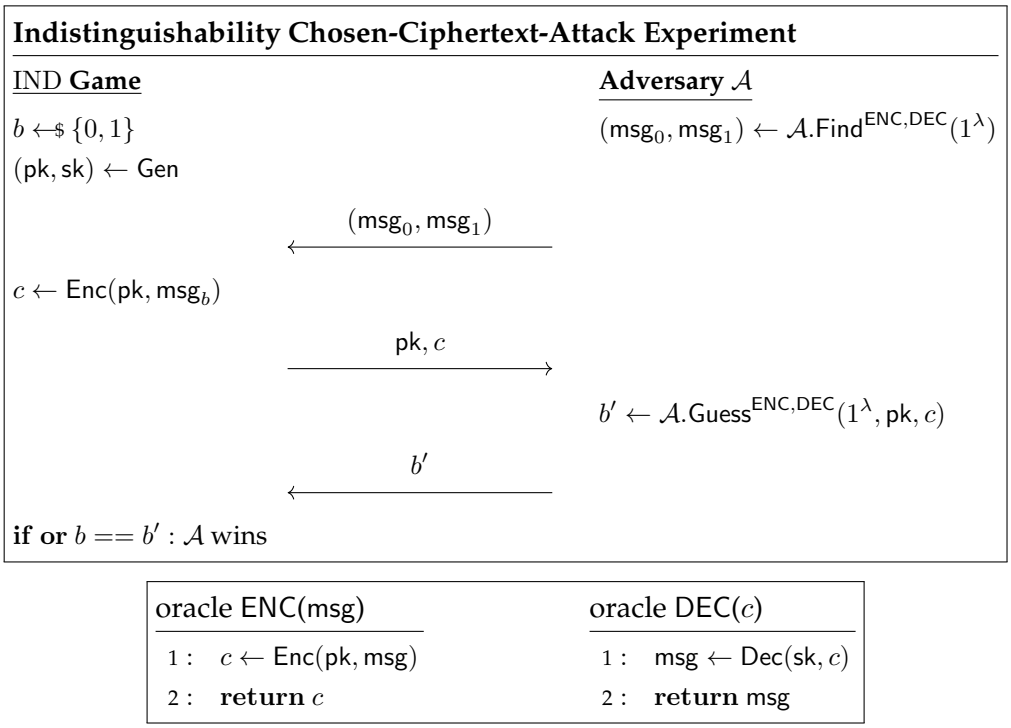


Figure 2.10: Depiction of the Public-Key Encryption-Scheme Indistinguishability-Chosen-Ciphertext-Attack Experiment, abbreviated to PKES-IND-CCA. In the experiment, an adversary $\mathcal{A}$ samples two messages msg_0 and msg_1 and sends them to the game. The game flips a coin and encrypts the according message msg_b. The encryption c and the public key pk are sent to the adversary. The adversary then needs to guess which message was encrypted. For this, $\mathcal{A}$ is provided with both an encryption and decryption oracle. If $\mathcal{A}$ guesses correctly, they win the game.

With this we have covered all necessary preliminaries for our thesis. On the following page, we conclude this chapter with a notation table (Figure 2.11). In the next chapter, we further provide a short technical overview of our work.

Name	Notation	Used in Chapters
Advantage	$\mathsf{Adv}^{\mathsf{game}}_{\mathcal{A},\Pi}(\lambda)$	2, 4, 6, 7
Adversary	$\mathcal{A}$	2, 4, 5, 6, 7
Cryptographic Primitive	Π	2
Insecurity	$\mathsf{InSec}^{\mathsf{game}}_{\Pi}(\lambda)$	2, 4, 6
Keys	$\mathsf{k}, \mathsf{ak}, \mathsf{ok}, \mathsf{pk}, \mathsf{sk}$	2, 4, 5, 6, 7
Negligible Function	$\mathsf{negl}(\lambda)$	2, 5, 6, 7
Polynomial Function	$\mathsf{poly}(\lambda)$	2, 4, 5, 6
Probabilistic Poly-Time	PPT	2, 6
Security Parameter	λ	2, 4, 5, 6, 7, A
Simulator	$\mathcal{S}$	2, 5
Channel	$\mathcal{C}$	2, 4, 6
Channel Message	msg	2, 4
History	h	2, 4
System State	σ	2, 4
Secret Message	sm	2, 4
Stegosystem	StS	2, 4, 6
Unreliability	$\mathsf{UnRel}_{\mathcal{A},\Pi}(\lambda)$	2, 4
Warden	W	2, 4, 6
Algorithm Substitution Attack	ASA	2, 4, 5, 7
Amalgamation Function	Am	2, 7
Subroutine	π	2, 7
Symmetric Encryption Scheme	SES	2, 4
Watchdog	Wa	2, 6, 7
Auditable Obfuscation Scheme	$\mathcal{AO}$	2, 4, 5, A
Auditor	Au	2, 4, A
Auxiliary Information	aux	2, 4
Correctness	ϕ_i	2, 4, 5
Implementation (set)	I $(\mathcal{I})$	2, 4, 5, A
Indistinguishability Obfuscation Scheme	$i\mathcal{O}$	2, 5, 6, 7, A
Malicious Obfuscation Scheme	$\mathcal{MO}$	2, 4, 5, A
Obfuscation Scheme	$\mathcal{O}$	2, 4, 5
Obfuscation	$\tilde{I}$	2, 4, 5
Program (set)	P $(\mathcal{P})$	2, 4, 5, A
Proof	ν	2, 5
Pseudorandom Generator	PRG (G)	2, 6, 7
Pseudorandom Function	PRF (F)	2, 6
Punctured Pseudorandom Function	PPRF (F)	2, 6, 7
Public Key Encryption Scheme	PKES $(\mathcal{E})$	2, 7

Figure 2.11: Table of Notations

3 Technical Overview

The scope of this thesis deals with two topics: the first part pertains to auditable obfuscation and its relation to subversion-resilience. The second part deals with subversion-resilient IND-CCA-secure encryption and how to achieve it.
In this chapter, we give a short technical overview of the thesis. We do this by presenting a broad introduction to the two sections, stating related research questions and a rough sketch of our results. We conclude the chapter with a visual representation of the thesis structure in Figure 3.1.

Auditable Obfuscation and its Relation to Subversion-Resilience

Auditable obfuscation was introduced by Banerjee and Galbraith in [BG23]. They provide a formal specification to verify and validate an obfuscation against a malicious obfuscator as well as an example construction for an auditable obfuscation scheme. In the formal specification of auditable obfuscation, the authors add a verifiability and soundness property to the standard definition of obfuscation. These two properties can be modeled as a verifying party, an auditor to the obfuscation scheme, leveraged as being able to observe and detect malicious modifications and embeddings in an obfuscated program.
The setting proposed by them thus closely resembles the subversion-resilience setting against algorithm substitution attacks. Analogously to the auditor modeling, subversion-resilience allows for the detection of algorithmic substitutions via an observing party, for which the term watchdog was coined by Russell et al. in [RTYZ16]. As auditable obfuscation still is a comparatively novel concept, there exist no works investigating its connection to subversion-resilience.
The first part of this thesis thus focuses on this resemblance, analyzes the relation between the two concepts as well as whether it only is superficial, due to the modeling of the problems, or the result of some underlying similarity. We pose this as our first research question:

Research Question 1 (On Auditable Obfuscation and Subversion-Resilience).
Can we show a deeper relation between the concept of auditable obfuscation and subversion-resilience?

We positively answer this question in Chapter 4.
We will do so by showing that malicious obfuscation, the inverse concept to auditable obfuscation, can be used to create a steganographic channel and vice versa, proving that the two settings are equivalent. We then combine this finding with the results of Berndt and Liśkiewicz [BL17], demonstrating that algorithm substitution attacks are similarly equivalent to the steganographic setting. We can then conclude that the inverse concept to algorithm substitution attacks, namely subversion-resilience, describes the same setting as auditable obfuscation.

Subversion-Resilient IND-CCA-Secure Encryption from Auditable Obfuscation

As discussed in the introduction, we not only want to show that auditable obfuscation is related to subversion-resilience, but also find a use case in which to successfully apply our result. Subversion-resilient IND-CCA-secure encryption provides such an application. While there are a multitude of ways to construct IND-CCA-secure encryption schemes, most of them do not fulfill the strict definition of subversion-resilience [BBD+23]. In general, no subversion-resilient construction for IND-CCA-secure asymmetric encryption has been found so far, making it an interesting goal for groundbreaking research.
To leverage our results from the first section, especially the fact that auditable obfuscation constitutes a subversion-resilient obfuscation scheme, we want to tackle IND-CCA-secure encryption from this view. Thankfully, building IND-CCA-secure encryption schemes from obfuscation has already been a research topic in recent years [SW14], mostly building up from the concept of indistinguishability obfuscation. The interesting further step forward now is to analyze whether we can leverage our results in combination with existing construction methods to achieve subversion-resilient IND-CCA-secure encryption.
One such promising construction is described in the 2014 paper [SW14] in which the authors, Sahai and Waters, leverage $i\mathcal{O}$ in combination with a secure PRG and punctured PRFs to prove IND-CCA security. As the construction is of low complexity, only relying on existing "$\oplus$" (*Xor*), "$\|$" (*Concatenation*) and "$=$" (*Equal*) operations beside the three mentioned cryptographic primitives, it proves to be a useful template for adaptation to subversion-resilience.
In this thesis, we will show that, given subversion-resilient cryptographic primitives, using the Sahai and Waters construction results in a subversion-resilient IND-CCA-secure encryption scheme. For our result to hold, we will further assume the three operations mentioned prior to be secure. A succinct reasoning for this can be found in the Preliminaries Subsection 2.2.3.
This puts our focus on the three cryptographic primitives, $i\mathcal{O}$, PRGs and PPRFs, for which we still need to show that they can be constructed in a subversion-resilient way. In prepa-

ration, we formulate three further research questions.
For the first primitive, $i\mathcal{O}$, we need to show that a subversion-resilient version of it exists. As we have a subversion-resilient obfuscation scheme in $\mathcal{AO}$, we need to show that we can construct $i\mathcal{O}$ from $\mathcal{AO}$. The resulting research question is:

Research Question 2 (On Subversion-Resilient Indistinguishability Obfuscation).
What is the relation between indistinguishability obfuscation and auditable obfuscation and can we leverage it to construct subversion-resilient indistinguishability obfuscation?

We partially answer this research question in Chapter 5.
In detail, we will show that $\mathcal{AO}$ implies $i\mathcal{O}$ and that we can thus leverage it to construct subversion-resilient indistinguishability obfuscation. For this, we will show that any valid $\mathcal{AO}$ scheme also fulfills all properties of an $i\mathcal{O}$ scheme while keeping its verifiability and soundness properties, which we can leverage for subversion-resilience.
The question if $i\mathcal{O}$ does or does not imply $\mathcal{AO}$ is not of relevance to our results. We leave this part of the research question open for future works to tackle. We will, however, provide a short discussion on it in the Appendix A.
For the second primitive, PRGs, the construction's root of randomness, we need to analyze if subversion-resilient constructions for it exist. Thus, we can formulate our next research question:

Research Question 3 (Subversion-Resilient Pseudorandom Generators).
Can we construct subversion-resilient PRGs?

We can affirm this research question in Chapter 6.
We do so by showing that PRGs are inherently subversion-resilient by leveraging the results of [RTYZ16], namely Lemma 2.13, which allows us to show that any functional subversion of a PRG must also be detectable.
Lastly, for PPRFs, which we require to argue the security of our construction, we also need to proved the existence of subversion-resilient constructions. This yields another simple, yet significant research question:

Research Question 4 (Subversion-Resilient Punctured Pseudorandom Functions).
Can we construct subversion-resilient punctured PRFs?

We answer the research question in Chapter 6.
In detail, we show that arbitrary punctured PRFs are not subversion-resilient, even if the underlying PRF is subversion-resilient, as the puncturing itself can be leveraged as a steganographic channel. However, we further show that using random puncturing, wherein the used randomness is not subverted, in combination with an additional rerandomization assumption fixes this problem and results in subversion-resilient punctured PRFs. This result is of no hindrance to our goal, as the Sahai and Waters construction already necessitates the use of randomly punctured PRFs [SW14].
Combining the results from these three research questions shows that all building blocks for the Sahai and Waters construction can be constructed subversion-resiliently. This leaves only the connecting step as the last possible attack vector. We formulate this as the final research question for this thesis:

Research Question 5 (On Subversion-Resilient IND-CCA-Secure Encryption).
Can we use the Sahai and Waters construction for CCA-secure encryption from indistinguishability obfuscation to construct a subversion-resilient CCA-secure encryption scheme?

We positively answer the research question in Chapter 7.
We do so by leveraging the trusted amalgamation with split-program model for construction, with which we can show that any successful subversion on the entire encryption scheme must also subvert one of its subroutines. Combining this with the results from the prior chapters then allows us to upper bound the advantage $\mathsf{Adv}^{\text{sub-res IND-CCA}}_{\mathcal{A},\mathcal{E}}(\lambda)$ by a negligible term. Additionally to this we will also need to prove IND-CCA security under subversion. We do this via the standard game hopping proof of Sahai and Waters [SW14], while sourcing the security from the subversion-resilience of its subroutines. Combining both proofs then shows that IND-CCA-secure encryption under subversion is not only possible, but also constructable.

Visualization

As an addition to the written technical overview of the following chapters, we want to provide an additional visual representation of the construction. With this depiction, we hope to give an additional intuition on the interaction between the different steps we make throughout this thesis. The graphic can be found in Figure 3.1.
The leftmost part of the picture depicts the conceptional equivalence between ASA, StS and $\mathcal{MO}$, which we show in Chapter 4. From this equivalence, we get that $\mathcal{AO}$, as the inverse to $\mathcal{MO}$, is subversion-resilient, which we mark through highlighting of its node.
From here on out, we visualize the results of Chapter 5 with the connection between the $\mathcal{AO}$ and $i\mathcal{O}$ nodes and the results of Chapter 6 with the highlighting of the PRG node as well as its connection to the PPRF node. Lastly, our results from Chapter 7 are depicted in the cross-connection between the $i\mathcal{O}$, PRG, PPRF and IND-CCA-Enc nodes. The Am node at the center is used to depict that we use the amalgamation model within this construction step.

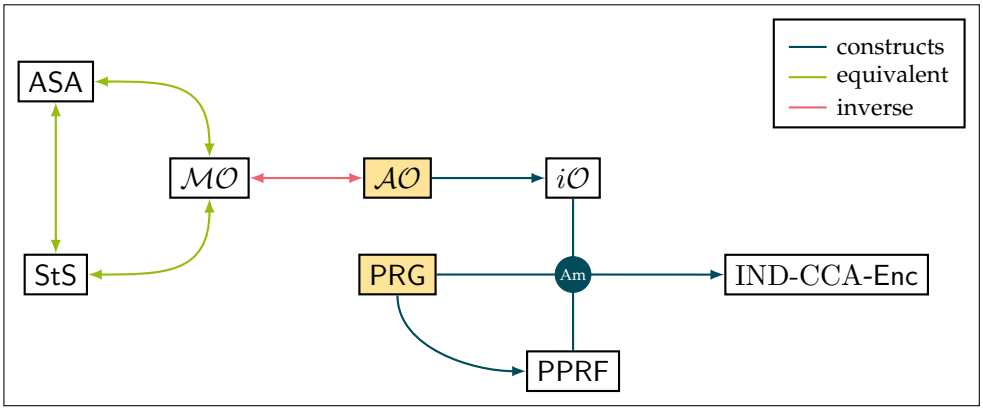
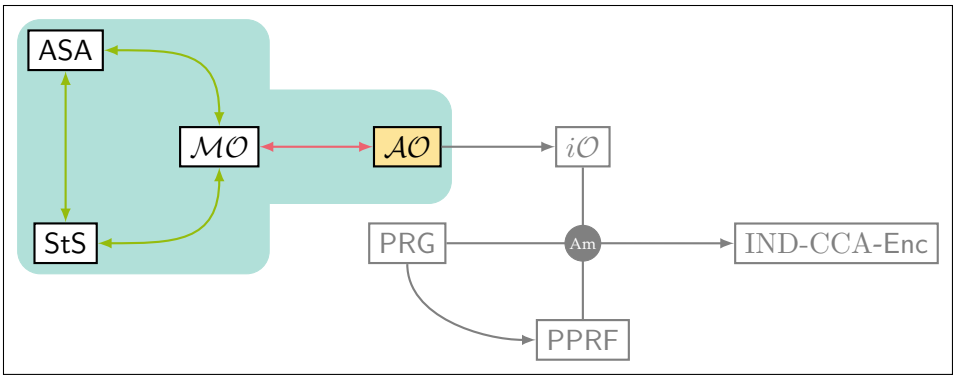


Figure 3.1: Visualization of how the different primitives throughout this thesis interact with each other. Primitives with green connections between them depict equivalent concepts, whereas red connections describe inverse concepts. Blue arrows are used to show that we can use the in-primitives to construct the out-primitive. Highlighted nodes are the inherently subversion-resilient primitives from which we source the property. As we will see in this thesis, any depicted construction preserves the subversion-resilience property.

Additionally, at the start of Chapters 4 through 7, we will provide a separate copy of Figure 3.1, highlighting which part of the construction is tackled in the respective chapter. In this way, we want to help the reader intuit at a glance the focus of the thesis at the point of reading.
We can now continue onward to the first research chapter, tackling subversion-resilient auditable obfuscation.

4 Subversion-Resilient Auditable Obfuscation



Auditable obfuscation is a novel concept introduced by Banerjee and Galbraith in [BG23]. The general idea behind the primitive is to *validate* that any given obfuscation does not differ from the underlying program. Stated differently, the primitive brings any generated obfuscation under *audit*, only accepting obfuscations that do not change the underlying program up to some correctness ϕ_i. Even if an individual obfuscator were to act maliciously, as long as the underlying obfuscation scheme is auditable, then this malicious behavior is either detected, or, with overwhelming probability, not exploitable.

As already stated in the technical overview, the setting thus closely resembles that of subversion-resilience. Subversion-resilient algorithms are described as working as intended even when subject to an algorithm substitution attack. Any existing subversion must either be detectable by a watchdog or, with overwhelming probability, not reliable.

In this chapter, we focus on this resemblance and show that, beyond the similarity of the settings, auditable obfuscation *is* just a subversion-resilient algorithm on the class of obfuscation schemes. To prove this equivalence, we closely follow the results from Berndt and Liśkiewicz in [BL17] and [Ber18].

Their work showed that algorithm substitution attacks and stegosystems describe the same setting. They accomplished this by proving that a stegosystem StS on a specific channel $\mathcal{C}_{\mathsf{SES}}$, described by a symmetric encryption scheme SES, can be leveraged to create an algorithm substitution attack ASA against the same SES and vice versa.

In this chapter, we will show that we can create a similar channel $\mathcal{C}_{\mathcal{O}}$, described by an honest obfuscation scheme $\mathcal{O}$, and use it analogously. We will prove that a stegosystem StS on this $\mathcal{C}_{\mathcal{O}}$ can be used to create a malicious obfuscation scheme $\mathcal{MO}$ over $\mathcal{O}$ and, similarly,

that we can leverage a malicious obfuscation scheme $\mathcal{MO}$ over $\mathcal{O}$ to create a stegosystem StS on $\mathcal{C}_{\mathcal{O}}$. By combining these two proofs, we will achieve the proof of equivalence mentioned above.
Firstly, we shall describe how to build channel $\mathcal{C}_{\mathcal{O}}$.

4.1 Obfuscation Schemes as a Steganographic Channel

The channel $\mathcal{C}_{\mathcal{O}}$ should describe a steganographic channel that is produced over a series of obfuscations. However, as our channel is described over $\mathcal{O}$, the channel must not only contain $\mathcal{O}$'s outputs, but also its inputs. As such, $\mathcal{C}_{\mathcal{O}}$ must contain an obfuscation key ok, a series of $\mathsf{poly}(\lambda)$ implementations I and, finally, a series of $\mathsf{poly}(\lambda)$ obfuscations $\tilde{I}$. Generally, we will define $\mathcal{C}_{\mathcal{O}}$ in such a way that it contains exactly one obfuscation for each provided implementation, providing a one to one mapping between implementations I_i and obfuscations $\tilde{I}_i$.
Now that we know what messages $\mathcal{C}_{\mathcal{O}}$ should contain, we need to discuss their order within the channel. The first ordering one might think of would be to fix the first message to the obfuscation key ok and then alternate between implementations I_i and their obfuscations $\tilde{I}_i$. Formally, a complete history with $\ell = \mathsf{poly}(\lambda)$ obfuscation calls for such a channel would thus be

$$\mathsf{h}_{\mathcal{C}_{\mathcal{O}}(\ell)}(1, 2\ell+1) \;=\; \mathsf{ok} \,||\, I_1 \,||\, \tilde{I}_1 \,||\, I_2 \,||\, \tilde{I}_2 \,||\, \ldots \,||\, I_\ell \,||\, \tilde{I}_\ell \quad .$$

However, such a channel causes a problem for our analysis. As we want to use $\mathcal{C}_{\mathcal{O}}$ to map between a stegosystem StS and malicious obfuscation scheme $\mathcal{MO}$, we only want the adversary in the steganographic setting to encode their messages into obfuscations. Otherwise, the adversary could encode messages *only* in plain implementations, which the adversary in the malicious obfuscation setting is never able to do.
Berndt and Liśkiewicz also ran into this exact problem when building their channel for ASA in [BL17]. They fixed this problem by splitting the channel into three separate states, an empty channel $\mathsf{h} = \emptyset$, a channel containing a key k and some plaintexts msg_1 to msg_i as well as a channel that contains a key k, all ℓ plaintexts msg_1 to msg_ℓ and some ciphertexts c_1 to c_i. For the final state, they further fixed the distribution of the ciphertexts in dependence on the provided key k and messages msg_1 to msg_ℓ. This allows them to then force the encoder in StS to only embed their sm in the ciphertexts.
We will now formally declare a similar channel description for $\mathcal{C}_{\mathcal{O}}$.

Let $\mathcal{O}_{\phi_i}$ be an obfuscator sufficing correctness ϕ_i for the family of implementations $\mathcal{I} = \{\mathcal{I}_\lambda\}$ with $\mathcal{I}_\lambda \subseteq \mathcal{P}_\lambda$, where $\mathcal{P}_\lambda$ is the set of all programs with parameterized inputs in λ, that takes some implementation $I \in \mathcal{I}$ as input and produces an output $\tilde{I} = \mathcal{O}.\mathsf{Obf}(I, \mathsf{ok})$. Then let $\mathcal{MO}_{\phi_i}$ be the malicious obfuscator scheme defined by $\mathcal{O}$. For both schemes, we will omit ϕ_i as long as it is not necessary for the argument being made. Further, let $\ell = \mathsf{poly}(\lambda)$. For $\mathcal{O}$, we define the channel $\mathcal{C}_\mathcal{O}(\ell)$, with which we want to capture the possible actions of $\mathcal{MO}$. As $\mathcal{MO}$ might be a piece of modular software used on programs from a third party, we can fix the instance of $\mathcal{O}$ over which $\mathcal{MO}$ is defined as well as the programs used. For this, we enforce that the first $\ell + 1$ messages sent via $\mathcal{C}_\mathcal{O}(\ell)$ must consist of the obfuscation key as well as the programs to be obfuscated.
As such:

$$\mathsf{h}_{\mathcal{C}_\mathcal{O}(\ell)}(1, \ell+1) \;=\; \mathsf{msg}_1 \,||\, \mathsf{msg}_2 \,||\, \ldots \,||\, \mathsf{msg}_{\ell+1} \;=\; \mathsf{ok} \,||\, I_1 \,||\, \ldots \,||\, I_\ell$$

where $\mathsf{ok} \leftarrow\!\$\; \mathcal{O}.\mathsf{Gen}(\cdot)$ and $I_i \leftarrow\!\$\; \mathcal{I}$.
After this fixing stage, the channel then consists of the obfuscations that may contain malicious embeddings. However, as the obfuscations must be indistinguishable to non-malicious obfuscations, we fix the distribution of obfuscations as follows:

$$\begin{aligned}\mathsf{h}_{\mathcal{C}_\mathcal{O}(\ell)}(\ell+2, 2\ell+1) \;&=\; \tilde{I}_1 \,||\, \tilde{I}_2 \,||\, \ldots \,||\, \tilde{I}_\ell \\ &\sim\; \mathcal{O}.\mathsf{Obf}(I_1, \mathsf{ok}) \,||\, \mathcal{O}.\mathsf{Obf}(I_2, \mathsf{ok}) \,||\, \ldots \,||\, \mathcal{O}.\mathsf{Obf}(I_\ell, \mathsf{ok})\end{aligned}$$

Given a specific implementation I_i, the output $\tilde{I_i}' \leftarrow \mathcal{MO}.\mathsf{Emb}(\mathsf{aux}, I_i, \mathsf{ok})$ is defined as *looking* like an honest obfuscation $\tilde{I}_i \leftarrow \mathcal{O}.\mathsf{Obf}(I_i, \mathsf{ok})$ while breaking the correctness class ϕ_i of $\mathcal{O}$. We denote the property of looking similar by $\tilde{I_i}' \sim \tilde{I}_i$ and the difference in correctness by $\tilde{I_i}' \not\sim_{\phi_i} \tilde{I}_i$. While we do not use this notation directly, it represents the underlying principle of $\mathcal{MO}.\mathsf{Trig}(\mathsf{aux}, \tilde{I})$ working as defined.
The entire channel $\mathcal{C}_\mathcal{O}$ is now defined by the histories and their respective distributions above. Generally, we will say a history h follows the distribution of $\mathcal{C}_\mathcal{O}$ if each respective position in h follows that position's distribution.
With this, $\mathcal{C}_\mathcal{O}$ has been formally defined. We continue our proof in the next section, showing that malicious obfuscation over an honest obfuscation scheme equals steganography.

4.2 Malicious Obfuscation against Obfuscation as Steganography

With $\mathcal{C} = \mathcal{C}_\mathcal{O}$ established, we now have to prove that one can translate between a malicious obfuscation scheme $\mathcal{MO}$ and stegosystem StS using $\mathcal{C}$. We do so in three distinct steps: Firstly, we prove that a given malicious obfuscation scheme over $\mathcal{O}$ implies a stegosys-

tem StS on $\mathcal{C}_{\mathcal{O}}$ of equal reliability. Secondly, we show the reverse, meaning that a given stegosystem StS on $\mathcal{C}_{\mathcal{O}}$ implies a malicious obfuscation scheme $\mathcal{MO}$ over $\mathcal{O}$. Lastly, we combine the results of the first two steps to prove that an auditable obfuscation scheme inherently is subversion-resilient.
The proofs for the first two parts will follow a similar structure. First, we provide a mapping between the respective PPTMs describing the primitives. We will then show that the insecurity of the original scheme upperbounds the insecurity of the newly created scheme while keeping the same unreliability. If we then assume the original scheme to be functional, it directly follows that the second scheme must also be functional as well.
We start by showing that malicious obfuscation implies steganography.

4.2.1 Malicious Obfuscation implies Steganography

As discussed above, the first step in showing that auditable obfuscation schemes are subversion-resilient will be presented in this subsection. We do so by showing that a functional malicious obfuscation scheme implies steganography. We formalize this mapping in the following theorem, proving it afterwards using the strategy outlined prior.

Theorem 4.1.
Assume $\mathcal{O}_{\phi_i}$ to be an obfuscation scheme with correctness ϕ_i and let $\mathcal{MO}_{\phi_i}$ be a malicious obfuscation scheme over $\mathcal{O}$. Further, let $\ell = \mathsf{poly}(\lambda)$. Then there exists a stegosystem StS on the channel $\mathcal{C} := \mathcal{C}_{\mathcal{O}}(\ell)$ determined by $\mathcal{O}$, such that

$$\begin{aligned} \mathsf{InSec}^{\mathsf{sts\text{-}cha\text{-}}\sigma}_{\mathsf{StS},\mathcal{C}}(\lambda) &\leq \mathsf{InSec}^{\mathsf{mal\text{-}obf}}_{\mathcal{MO},\mathcal{O}}(\lambda) \quad \textit{and} \\ \mathsf{UnRel}_{\mathsf{StS},\mathcal{C}}(\lambda) &= \mathsf{UnRel}_{\mathcal{MO},\mathcal{O}}(\lambda) \quad . \end{aligned}$$

Proof. Let $\mathcal{O}$ be an obfuscation scheme over the family of implementations $\mathcal{I} = \{\mathcal{I}_\lambda\}$ with $\mathcal{I}_\lambda \subseteq \mathcal{P}_\lambda$ and $\mathcal{MO}$ a malicious obfuscation scheme defined over $\mathcal{O}$ parameterized in λ. Further, let $\ell = \mathsf{poly}(\lambda)$ and let us assume that we can sample elements from $\mathcal{I}$.
We will now construct the stegosystem StS on $\mathcal{C}_{\mathcal{O}}$ from $\mathcal{MO}$. For this, we need to show how to construct the three PPTMs defining StS; StS.Gen, StS.Enc and StS.Dec.

StS.Gen
We first look at the generator function StS.Gen, which can simply be built by simulating the $\mathcal{MO}$ generator function $\mathcal{MO}$.Gen. We interpret its output aux as the message key mk for the stegosystem.

StS.Enc

For StS.Enc, we need to show that we can create a valid message history for $\mathcal{C}_{\mathcal{O}}$ while also encoding some message sm. We assume that $\mathsf{sm} \in \{0,1\}^{\ell}$.

We will construct a valid history h for $\mathcal{C}_{\mathcal{O}}$ iteratively based on the history of messages sent prior:

- If $\mathsf{h} = \emptyset$, then $\mathcal{C}_{\mathcal{O}}$ forces a key $\mathsf{ok} \sim \mathcal{O}.\mathsf{Gen}(\cdot)$ as its next message msg_1. The encryption function can simply sample such a random key ok by running the generation algorithm of $\mathcal{O}$ and outputting it.
- If $\mathsf{h} = \mathsf{ok} \mid\mid I_1 \mid\mid I_2 \mid\mid \ldots \mid\mid I_s$ with $0 \leq s \leq \ell - 1$, then channel $\mathcal{C}_{\mathcal{O}}$ forces an implementation $I_i \leftarrow\!\!\$\ \mathcal{I}$ as its next message. As the family of implementations $\mathcal{I}$ is public, StS.Enc can simply sample such a random implementation I_i as the next message.
- If $\mathsf{h} = \mathsf{ok} \mid\mid I_1 \mid\mid I_2 \mid\mid \ldots \mid\mid I_\ell \mid\mid \tilde{I}_1 \mid\mid \ldots \mid\mid \tilde{I}_s$ with $0 \leq s \leq \ell - 1$, then $\mathcal{C}_{\mathcal{O}}$ forces an obfuscation $\tilde{I}_{s+1} = \mathcal{O}.\mathsf{Obf}(I_{s+1}, \mathsf{ok})$ as its next message. By assumption, we know that $\mathcal{MO}$ is a functioning malicious obfuscator on $\mathcal{O}$. As such, we know that

  $$\mathcal{MO}.\mathsf{Emb}(\mathsf{aux}, I_i, \mathsf{ok}) \sim \mathcal{O}.\mathsf{Obf}(I_i, \mathsf{ok}) \quad .$$

  StS.Enc will now use this similarity to encode sm. In the current step $s+1$, $\mathsf{sm}[s+1]$ is encoded by either simulating $\mathcal{MO}$.Emb to send a 1 or $\mathcal{O}$.Obf to send a 0. This means StS.Enc generates the next message as

  $$\tilde{I}_{s+1} = \begin{cases} \mathcal{MO}.\mathsf{Emb}(\mathsf{mk}, I_{s+1}, \mathsf{ok}) & , \text{if } \mathsf{sm}[s+1] = 1 \\ \mathcal{O}.\mathsf{Obf}(I_{s+1}, \mathsf{ok}) & , \text{if } \mathsf{sm}[s+1] = 0 \end{cases} \quad .$$

  The resulting $\tilde{I}_{s+1}$ is then output as the next message.

Following these rules results in a valid channel history, while also encoding sm.

StS.Dec

Lastly, we need to construct StS.Dec that receives the message stream output

$$(\mathsf{msg}_1, \mathsf{msg}_2, \ldots, \mathsf{msg}_{2\ell+1})$$

from StS.Enc and tries to reconstruct sm. For this, we can use the trigger of the malicious obfuscator, $\mathcal{MO}$.Trig. When the trigger functionality is called as $\mathcal{MO}.\mathsf{Trig}(\mathsf{aux}, \tilde{I})$, the function tries to execute an embedding in the provided obfuscation $\tilde{I}$. If the provided obfuscation contains a malicious embedding, $\mathcal{MO}$.Trig outputs a 1, otherwise a 0. If we provide more than one obfuscation, we assume that $\mathcal{MO}$.Trig is sequentially run on all

obfuscations.
As only the messages $(\mathsf{msg}_{\ell+2}, \ldots, \mathsf{msg}_{2\ell+1})$ are allowed to contain malicious embeddings, StS.Dec can now use the output from $\mathcal{MO}.\mathsf{Trig}(\mathsf{mk}, \mathsf{msg}_{\ell+2}, \ldots, \mathsf{msg}_{2\ell+1})$ as its recovery for sm^*.

With this, we have shown that all three PPTMs that define StS can be constructed from $\mathcal{MO}$. It remains to show that the unreliability and insecurity of the constructed stegosystem StS suffice the theorem. We first analyze the security of the system by showing that a warden can be used on StS to create an auditor against $\mathcal{MO}$.
Let W be a warden against StS on $\mathcal{C} = \mathcal{C}_{\mathcal{O}}$ with maximal advantage, meaning

$$\mathsf{Adv}^{\mathrm{sts-cha}-\sigma}_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda) = \mathsf{InSec}^{\mathsf{sts-cha}-\sigma}_{\mathsf{StS},\mathcal{C}}(\lambda) \quad .$$

We will construct an auditor Au against the malicious obfuscation scheme $\mathcal{MO}$ with the same advantage as W, meaning that

$$\mathsf{Adv}^{\mathrm{mal-obf}}_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda) = \mathsf{Adv}^{\mathrm{sts-cha}-\sigma}_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda) \quad .$$

This will prove that

$$\mathsf{InSec}^{\mathsf{sts-cha}-\sigma}_{\mathsf{StS},\mathcal{C}}(\lambda) \leq \mathsf{InSec}^{\mathsf{mal-obf}}_{\mathcal{MO},\mathcal{O}}(\lambda) \quad .$$

We construct the auditor Au as follows: When Au is run on input 1^λ, they simulate the warden W. As Au acts as the game to W, they need to answer all of the warden's oracle queries. W is given an encryption oracle Enc, a channel oracle $\mathcal{C}$ and a challenge oracle CH. We need to prove that we can perfectly simulate all three of them.

W.Enc
Let us first look at the encryption oracle. Whenever W makes a query to $\mathsf{W.Enc}(\mathsf{sm}, \mathsf{h}, \sigma)$, Au answers depending on the current state of h:

- If $\mathsf{h} = \emptyset$, then Au returns $\mathsf{ok} \leftarrow\!\$\ \mathcal{O}.\mathsf{Gen}(1^\lambda)$.
- If $\mathsf{h} = \mathsf{ok} \mid\mid I_1 \mid\mid I_2 \mid\mid \ldots \mid\mid I_s$ with $0 \leq s \leq \ell - 1$, a random implementation $I_i \leftarrow\!\$\ \mathcal{I}$ is chosen and output by Au.
- If $\mathsf{h} = \mathsf{ok} \mid\mid I_1 \mid\mid I_2 \mid\mid \ldots \mid\mid I_\ell \mid\mid \tilde{I}_1 \mid\mid \ldots \mid\mid \tilde{I}_s$ with $0 \leq s \leq \ell - 1$, Au looks at $\mathsf{sm}[s+1]$ and answers depending on its bit value. If the bit is 0, Au calculates $\mathcal{O}.\mathsf{Obf}(I_{s+1}, \mathsf{ok})$ and returns it. Otherwise, the auditor calls its own embedding oracle $\mathsf{Au.Emb}(I_{s+1}, \mathsf{ok}, \sigma)$ and returns its output.

This constitutes a perfect simulation of the encryption oracle, as the oracle simulation exactly matches the defined behavior of our prior construction.

W.$\mathcal{C}$

Next we look at the channel oracle. Whenever W makes a query to $\mathcal{C}$ with a provided history h, Au answers, again depending on the state of h:

- If $\mathsf{h} = \emptyset$, then Au generates a key ok via the obfuscator's generation function $\mathcal{O}.\mathsf{Gen}(1^\lambda)$ and returns it.
- If $\mathsf{h} = \mathsf{ok} \mid\mid I_1 \mid\mid I_2 \mid\mid \ldots \mid\mid I_s$ with $0 \leq s \leq \ell - 1$, then Au samples $I_i \leftarrow\!\!\$\ \mathcal{I}$ and outputs it.
- If $\mathsf{h} = \mathsf{ok} \mid\mid I_1 \mid\mid I_2 \mid\mid \ldots \mid\mid I_\ell \mid\mid \tilde{I}_1 \mid\mid \ldots \mid\mid \tilde{I}_s$ with $0 \leq s \leq \ell - 1$, Au evaluates $\tilde{I}_{s+1} \leftarrow \mathcal{O}.\mathsf{Obf}(I_{s+1}, \mathsf{ok})$ and returns $\tilde{I}_{s+1}$.

As the auditor's simulation exactly follows the description of the channel distribution $\mathcal{C}$ from Section 4.1, this constitutes a perfect simulation.

W.CH

Lastly, we handle the challenge oracle. Whenever warden W makes a query to its challenge oracle with $\mathsf{CH}(\mathsf{sm}, \mathsf{h}, \sigma)$, Au simulates the output as follows:

- If $\mathsf{h} = \emptyset$, then Au simply samples $\mathsf{ok} \leftarrow\!\!\$\ \mathcal{O}.\mathsf{Gen}(1^\lambda)$ and returns it.
- If $\mathsf{h} = \mathsf{ok} \mid\mid I_1 \mid\mid I_2 \mid\mid \ldots \mid\mid I_s$ with $0 \leq s \leq \ell - 1$, Au chooses a random implementation $I_i \leftarrow\!\!\$\ \mathcal{I}$ and returns it.
- If $\mathsf{h} = \mathsf{ok} \mid\mid I_1 \mid\mid I_2 \mid\mid \ldots \mid\mid I_\ell \mid\mid \tilde{I}_1 \mid\mid \ldots \mid\mid \tilde{I}_s$ with $0 \leq s \leq \ell - 1$, then Au queries its own challenge oracle $\mathsf{Au.CH}(\mathsf{sm}, \mathsf{ok}, I_{s+1}, \sigma)$ and passes the result onto the channel.

If the auditor's challenge is a non-malicious obfuscation, meaning Au.CH is identically distributed to $\mathcal{O}.\mathsf{Obf}$, then the simulated warden challenge oracle is similarly identically distributed to $\mathcal{C}$. Thus,

$$\Pr[\mathsf{MO\text{-}CEA\text{-}Dist}\text{-}\sigma_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda)\langle b = 0\rangle = 1] = \Pr[\mathsf{StS\text{-}CHA\text{-}Dist}\text{-}\sigma_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda)\langle b = 0\rangle = 1] \,.$$

Correspondingly, if Au's challenge is a malicious obfuscation, meaning Au.CH returns $\mathcal{MO}.\mathsf{Emb}$, then Ws challenge oracle is identically distributed to its encryption oracle StS.Enc. Thus,

$$\Pr[\mathsf{MO\text{-}CEA\text{-}Dist}\text{-}\sigma_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda)\langle b = 1\rangle = 1] = \Pr[\mathsf{StS\text{-}CHA\text{-}Dist}\text{-}\sigma_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda)\langle b = 1\rangle = 1] \,.$$

The oracle thus is perfectly simulated, meaning that Au can always win when W wins. This further implies that the probabilities of success are equal in both the honest and malicious case. We can now insert this into the definition of the respective game advan-

tages to show that

$$\begin{aligned}\mathsf{Adv}^{\mathrm{mal-obf}}_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda) &= |\Pr[\mathsf{MO\text{-}CEA\text{-}Dist}\text{-}\sigma_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda) = 1] - 0.5| \\ &= |\Pr[\mathsf{SS\text{-}CHA\text{-}Dist}\text{-}\sigma_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda) = 1] - 0.5| \\ &= \mathsf{Adv}^{\mathrm{sts-cha}-\sigma}_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda) \quad .\end{aligned}$$

With this, we have demonstrated that we satisfy the stated security requirement of the system, leaving the reliability of the stegosystem left to analyze.

StS.Dec works by testing position-wise whether the obfuscated implementation is malicious or not. For a 0 in our message, we must not detect a non-malicious obfuscation as malicious. For the extractor, this probability should be equal to the success probability of $\mathcal{MO}$. For a 1 in our message, we must detect a malicious obfuscation as malicious. For the extractor, this probability should, again, be equal to $\mathcal{MO}$'s probability of success.
The reliability of the entire StS scheme thus is equal to the success probability of $\mathcal{MO}$, or inversely

$$\mathsf{UnRel}_{\mathsf{StS},\mathcal{C}}(\lambda) \;=\; \mathsf{UnRel}_{\mathcal{MO},\mathcal{O}}(\lambda) \quad . \qquad \square$$

We managed to show that a functioning malicious obfuscation scheme $\mathcal{MO}$ over $\mathcal{O}$ implies the existence of a functional stegosystem StS on $\mathcal{C}_{\mathcal{O}}$. We move on to the second step, showing that a functional stegosystem StS on $\mathcal{C}_{\mathcal{O}}$ also implies the existence of a functioning malicious obfuscation scheme $\mathcal{MO}$ over $\mathcal{O}$.

4.2.2 Steganography implies Malicious Obfuscation

Similarly to the last subsection, we want to prove that steganography on a certain type of channel implies malicious obfuscation. For this, we show that a functional stegosystem StS on $\mathcal{C}_{\mathcal{O}}$ can be leveraged to construct a malicious obfsucation scheme $\mathcal{MO}$ over $\mathcal{O}$. We formalize the statement in the following theorem and prove it afterwards using the outlined strategy.

Theorem 4.2.
Assume $\mathcal{O}_{\phi_i}$ to be an obfuscation scheme with correctness ϕ_i and let $\ell = \mathsf{poly}(\lambda)$. Further, let StS *be a stegosystem on the channel $\mathcal{C} := \mathcal{C}_{\mathcal{O}}(\ell)$ determined by $\mathcal{O}$. Then, there exists a malicious obfuscation scheme $\mathcal{MO}_{\phi_i}$ over $\mathcal{O}$ such that*

$$\begin{aligned}\mathsf{InSec}^{\mathsf{mal\text{-}obf}}_{\mathcal{MO},\mathcal{O}}(\lambda) \;&\leq\; \mathsf{InSec}^{\mathsf{sts\text{-}cha}\text{-}\sigma}_{\mathsf{StS},\mathcal{C}}(\lambda) \quad \textit{and} \\ \mathsf{UnRel}_{\mathcal{MO},\mathcal{O}}(\lambda) \;&=\; \mathsf{UnRel}_{\mathsf{StS},\mathcal{C}}(\lambda) \quad .\end{aligned}$$

Proof. Let $\mathcal{O}$ be an obfuscation scheme over the family of implementations $\mathcal{I} = \{\mathcal{I}_\lambda\}$ with $\mathcal{I}_\lambda \subseteq \mathcal{P}_\lambda$ and let StS be a stegosystem on the channel $\mathcal{C}$ determined by $\mathcal{O}$.
Further let $\ell = \mathsf{poly}(\lambda)$.
We will now construct the malicious obfuscation scheme $\mathcal{MO}$ over $\mathcal{O}$ from StS. For this, we need to show how to construct the three PPTMs defining $\mathcal{MO}$: $\mathcal{MO}$.Gen, $\mathcal{MO}$.Emb and $\mathcal{MO}$.Trig.

$\mathcal{MO}$.Gen
The generator algorithm can be constructed by simply simulating StS.Gen. We interpret the resulting mk as our auxiliary information aux.

$\mathcal{MO}$.Emb
The embedding function is provided some auxiliary information aux, an implementation I and an obfuscation key $\mathsf{ok} \in \mathrm{Supp}(\mathcal{O}.\mathsf{Gen}(\lambda))$ and produces an output $\tilde{I}'$, which is an obfuscation of I with some malicious embedding. When trying to recreate $\mathcal{MO}$.Emb with StS, we need to keep in mind that all steganographic functionality of StS is only defined over $\mathcal{C}$. As such, we need to create a valid history, such that StS.Enc can encode a steganographic message sm.
StS.Enc is allowed to encode sm after the key to the underlying scheme $\mathcal{O}$ and all base-implementations $I_1, \ldots, I_\ell$ are fixed. To build $\mathcal{MO}$.Emb, we thus need StS to run on a history $\mathsf{h}(\mathsf{ok}, I) = \mathsf{ok} \,\|\, I \,\|\, I \,\|\, \ldots \,\|\, I$ that consists of the obfuscation key ok and ℓ copies of implementation I. The algorithm StS.Enc then is allowed to execute and encode sm into the obfuscations $\tilde{I}_1, \ldots, \tilde{I}_\ell$ that occupy the messages $\mathsf{msg}_{\ell+2}$ to $\mathsf{msg}_{2\ell+1}$ on $\mathcal{C}$. We can now return this series of obfuscations as output of $\mathcal{MO}.\mathsf{Emb}(\mathsf{aux}, I, \mathsf{ok})$.

$\mathcal{MO}$.Trig
For a given input series of obfuscations $\tilde{I}_1, \ldots, \tilde{I}_\ell$, we can simply run StS.Dec on it to produce a recovered message sm'. If it holds that $\mathsf{sm}' = \mathsf{sm}$, we return 1, else 0. With this, we have shown that all three PPTMs defining $\mathcal{MO}$ can be constructed from a stegosystem StS.

We have yet to show that the unreliability and insecurity of the constructed malicious obfuscation scheme $\mathcal{MO}$ suffice the theorem. We first analyze the security of the scheme. Let Au be an auditor against $\mathcal{MO}$ on $\mathcal{O}$ with maximal advantage, meaning

$$\mathsf{Adv}^{\mathrm{mal-obf}}_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda) = \mathsf{InSec}^{\mathsf{mal-obf}}_{\mathcal{MO},\mathcal{O}}(\lambda) \quad .$$

We now construct a warden W against the stegosystem StS on $\mathcal{C} = \mathcal{C}_\mathcal{O}$ with the same

advantage as Au, meaning that

$$\mathsf{Adv}^{\mathrm{sts-cha}-\sigma}_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda) = \mathsf{Adv}^{\mathrm{mal-obf}}_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda) \quad .$$

This proves that

$$\mathsf{InSec}^{\mathsf{mal}-\mathsf{obf}}_{\mathcal{MO},\mathcal{O}}(\lambda) \leq \mathsf{InSec}^{\mathsf{sts}-\mathsf{cha}-\sigma}_{\mathsf{StS},\mathcal{C}}(\lambda) \quad .$$

We construct the warden W as follows: When W is run on input 1^λ, they simulate the auditor Au. As W acts as the game to Au, they need to answer all of the auditor's oracle queries. Au is given an embedding oracle Emb and a challenge oracle CH. We need to prove that we can perfectly simulate both.

Au.Emb
Whenever the auditor Au makes a query to the embedding oracle $\mathsf{Au.Emb}(I, \mathsf{ok}, \sigma)$, warden W acts as follows: First, W builds the history $\mathsf{h}(\mathsf{ok}, I) = \mathsf{ok} \,\|\, I \,\|\, \ldots \,\|\, I$. Afterwards, they choose an arbitrary value $\mathsf{sm} \in \{0,1\}^\ell$. With this, W then queries their own encryption oracle $\mathsf{W.Enc}(\mathsf{sm}, \mathsf{h}(\mathsf{ok}, I), \sigma)$. The result is output to answer Au's oracle request.
By design of the $\mathcal{MO}$ scheme above, this simulates the oracle's expected outputs perfectly.

Au.CH
Whenever Au makes a query to their channel oracle with $\mathsf{CH}(I, \mathsf{ok}, \sigma)$, W simulates the result as follows: First, W builds the history $\mathsf{h}(\mathsf{ok}, I)$ analogously to the procedure above and chooses an arbitrary message sm. Then, it queries its own challenge oracle $\mathsf{W.CH}(\mathsf{sm}, \mathsf{h}(\mathsf{ok}, I), \sigma)$ and passes its output to Au.
If the warden's challenge is a channel history with no encoded steganographic message sm, meaning that W.CH is identically distributed to $\mathcal{C}$, the challenge oracle of the simulated auditor is similarly identically distributed to the output of $\mathcal{O}$.Obf. Thus,

$$\Pr[\mathsf{StS\text{-}CHA\text{-}Dist}\text{-}\sigma_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda)\langle b = 0\rangle = 1] = \Pr[\mathsf{MO\text{-}CEA\text{-}Dist}\text{-}\sigma_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda)\langle b = 0\rangle = 1] \; .$$

Correspondingly, if the warden's challenge contains a steganographic message encrypted in the channel history, meaning that W.CH outputs the result of StS.Enc, then Au's challenge oracle is identically distributed to $\mathcal{MO}$.Emb. Thus,

$$\Pr[\mathsf{StS\text{-}CHA\text{-}Dist}\text{-}\sigma_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda)\langle b = 1\rangle = 1] = \Pr[\mathsf{MO\text{-}CEA\text{-}Dist}\text{-}\sigma_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda)\langle b = 1\rangle = 1] \; .$$

Using this result, we can gather that the oracle Au.CH is perfectly simulated, meaning that W can always win when Au wins, proving that the success probabilities of both

games are equivalent. We can now insert this into the respective game advantages, yielding

$$\begin{aligned}\mathsf{Adv}^{\mathrm{ss-cha-}\sigma}_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda) &= |\Pr[\text{SS-CHA-Dist-}\sigma_{\mathsf{W},\mathsf{StS},\mathcal{C}}(\lambda) = 1] - 0.5| \\ &= |\Pr[\text{MO-CEA-Dist-}\sigma_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda) = 1] - 0.5| \\ &= \mathsf{Adv}^{\mathrm{mal-obf}}_{\mathsf{Au},\mathcal{MO},\mathcal{O}}(\lambda) \quad .\end{aligned}$$

This proves that we satisfy the stated security requirement of the system. We now have to prove that we also fulfill the reliability requirement.

However, as the output of $\mathcal{MO}.\mathsf{Trig}$ just calls $\mathsf{StS.Dec}$ and decides its return value based on the result of the decryption, it must hold that $\mathcal{MO}.\mathsf{Trig}$ functions exactly when $\mathsf{StS.Dec}$ works correctly. Thus, it follows that

$$\mathsf{UnRel}_{\mathcal{MO},\mathcal{O}}(\lambda) = \mathsf{UnRel}_{\mathsf{StS},\mathcal{C}}(\lambda) \quad . \qquad \square$$

We also managed to show that a functioning stegosystem StS on $\mathcal{C}_{\mathcal{O}}$ implies the existence of a functional malicious obfuscator $\mathcal{MO}$ over $\mathcal{O}$. However, we only managed to do so under the assumption that a series of obfuscations is a valid output for an obfuscator $\mathcal{O}$. While it certainly is possible, that better encodings for sm would fix this assumption, we argue that the assumption is sound. For example, each obfuscation $\tilde{I}_i$ for an implementation I could correspond to an obfuscation for a distinct operating system. In this case, $\mathcal{O}$ would provide an obfuscation for (*up to*) $\mathsf{poly}(\lambda)$ operating systems at once, which a user would arguably consider a net positive instead of suspicious.
Finally, we need to combine the result of this subsection with the result of Subsection 4.2.1.

4.2.3 Auditable Obfuscation is Subversion-Resilient

We established that we can respectively translate between a functional malicious obfuscator $\mathcal{MO}$ and a stegosystem StS using a channel $\mathcal{C}_{\mathcal{O}}$ that follows our definition in Section 4.1. This was done to try and prove that auditable obfuscation, the inverse problem to malicious obfuscation, inherently is subversion-resilient.
As the inverse problem to $\mathcal{MO}$, we defined $\mathcal{AO}$ as describing any malicious obfuscation only being able to work with some negligible advantage, resulting in its reliability being negligible. As we showed that we can upperbound the reliability of any stegosystem StS with the reliability of a malicious obfuscator $\mathcal{MO}$, this implies that there can be no effective stegosystem over $\mathcal{AO}$.
We shall now combine our results with the prior research of Berndt and Liśkiewicz in [BL17] to formally prove the following statement:

Theorem 4.3 (Subversion-Resilient Auditable Obfuscation).
Any Auditable Obfuscation scheme $\mathcal{AO}$ inherently is subversion-resilient.

Proof. We proved both Theorem 4.1 and Theorem 4.2 in Section 4.2. The two theorems respectively show that we can instantiate malicious obfuscation $\mathcal{MO}$ to create a stegosystem StS and vice versa. Formally it thus holds that

$$\mathcal{MO} \leftrightarrow \mathsf{StS} \quad .$$

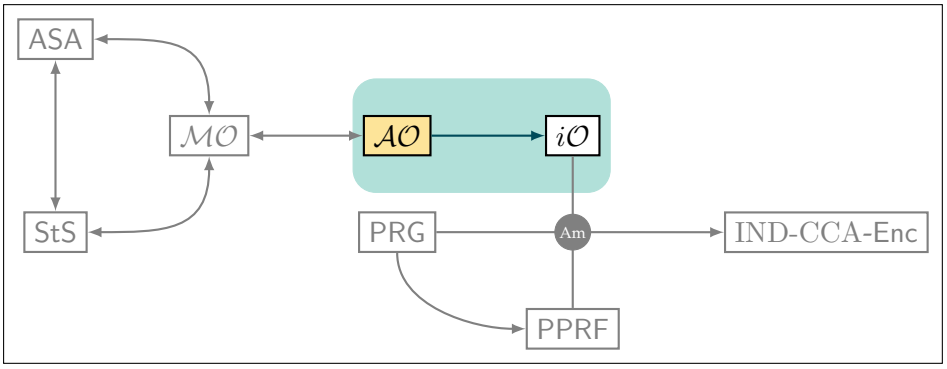
As $\mathcal{AO}$ implies the absence of a functional malicious obfuscation scheme, it also implies the absence of a reliable stegosystem. We combine this result with the findings of Berndt and Liśkiewicz in [BL17, Ber18] showing that algorithm substitution attacks are equivalent to stegosystems as well. Formally, it thus holds that

$$\mathcal{MO} \leftrightarrow \mathsf{StS} \leftrightarrow \mathsf{ASA} \quad .$$

$\mathcal{AO}$ thus also implies the absence of a functional algorithm substitution attack. By the same argument, the inverse implication holds true as well. $\mathcal{AO}$ and subversion-resilience thus describe the same concept in different settings. □

With this, we managed to achieve our goal for this chapter, showing that auditable obfuscation is subversion-resilient. While this proof alone constitutes an interesting result, we also want to try providing a first application of it. For this, we will use the following chapters to show that subversion-resilient auditable obfuscation allows us to achieve subversion-resilient IND-CCA-secure encryption. This process begins in the next chapter by showing how to create subversion-resilient indistinguishability obfuscation from $\mathcal{AO}$.

5 From Auditable Obfuscation to Subversion-Resilient Indistinguishability Obfuscation



In the prior chapter, we analyzed auditable obfuscation ($\mathcal{AO}$) and how it relates to subversion-resilience. We showed that any valid $\mathcal{AO}$ scheme must inherently suffice the definition of subversion-resilience due to its inverse problem, malicious obfuscation ($\mathcal{MO}$) being equivalent to a steganographic attack and thereby also equivalent to algorithm substitution attacks (ASA). These and their countermeasure of subversion-resilience have become a hot topic of research in recent years, largely due to some real world examples substantiating them as a realistic attack scenario [BPR14, DFP15].

The majority of ASA-related research focuses on cryptographic primitives, as they constitute a prime target. However, most implementations of cryptographic primitives are not inherently primed against algorithm substitution attacks due to their complexity. As such, plenty of research has been focused on building subversion-resilient constructions from the few primitives providing inherent subversion-resilience, such as unique ciphertext encryption [BPR14, DFP15], unique signature schemes [AMV20], and weak-PRFs [BBD+23, BBC24].

With auditable obfuscation, we have extended this small subsection of cryptographic primitives by another entry. However, as $\mathcal{AO}$ still is a comparatively novel concept, it has not undergone any rigorous scientific analysis, meaning its relation to other cryptographic primitives being mostly unknown, limiting its current use. We therefore aim to relate $\mathcal{AO}$ to a more well-understood cryptographic primitive.

We shall do so by showing that an auditable obfuscation scheme directly implies the existence of an indistinguishability obfuscation scheme. Furthermore, we will show that this

construction even is subversion-resilient. Indistinguishability obfuscation ($i\mathcal{O}$) has been the focus of research for a while [BGI+01] and its relation to other cryptographic primitives is well understood, allowing a subversion-resilient construction of it to be leveraged in a multitude of ways. We will show one such application in Chapter 7, where we leverage this chapter's results in constructing the first subversion-resilient IND-CCA-secure encryption. In Section 8.2, we will further provide a short discussion on other possible utilizations.

5.1 Auditable Obfuscation implies Indistinguishability Obfuscation

Auditable obfuscation describes a verifiable, sound obfuscation scheme on some family of programs $\mathcal{P}$ and its implementations $\mathcal{I}$. In general, $\mathcal{P}$ contains a multitude of programs, but the set could be as small as a single program. The resulting obfuscation should then indistinguishably hide which implementation was used while still providing the user with enough information to *audit* the scheme, verifying that some specified level of correctness, ϕ_i, is upheld.
Indistinguishability obfuscation, on the other hand, describes an obfuscation scheme defined for a single program P and its implementations $\mathcal{I}_P$. Its resulting obfuscation only hides which implementation was originally entered into the scheme.
When comparing the two descriptions, one can come to the conclusion that auditable obfuscation describes a more generalized scheme than indistinguishability obfuscation, simply extending its input realm to cases of $|\mathcal{P}| > 1$ and adding auditability. Vice versa, indistinguishability obfuscation can be described as an auditable obfuscation scheme with a reduced input realm $|\mathcal{P}| = 1$ that additionally drops the soundness and verifiability properties.
As we can describe $i\mathcal{O}$ by *weakening* or removing properties of $\mathcal{AO}$, we seemingly can instantiate an $i\mathcal{O}$ scheme from an $\mathcal{AO}$ scheme. We do so in the following theorem:

Theorem 5.1.
Let $\mathcal{P}_\lambda$ be the set of all programs with inputs parameterized in $\lambda \in \mathbb{N}$.
If $\mathcal{O}$ is an auditable obfuscation scheme of correctness ϕ_i defined over a family of polynomial-size program implementations $\mathcal{I} = \{\mathcal{I}_\lambda\}$ with $\mathcal{I}_\lambda \subseteq \mathcal{P}_\lambda$, then $\mathcal{O}$ also is an indistinguishability obfuscation scheme of correctness ϕ_i defined over a family of polynomial-size program implementations $\mathcal{I}' = \{P\}$ with $P \in \mathcal{I}$.

Proof. Let $\mathcal{O}$ be a tuple of PPTMs sufficing the definition of an auditable obfuscation scheme of correctness ϕ_i, defined over a family of polynomial-size program implementations $\mathcal{I} = \{\mathcal{I}_\lambda\}$ with $\mathcal{I}_\lambda \subseteq \mathcal{P}_\lambda$.
We will now constructively show that for any choice of $P \in \mathcal{I}$, $\mathcal{O}$ also is an indistinguishability obfuscation scheme over the family of polynomial-size program implementations $\mathcal{I}' = \{P\}$. To achieve this, we show how to construct $\mathcal{I}'$ from $\mathcal{I}$, why $\mathcal{I}'$ must contain all poly-sized implementations I of P, and that $\mathcal{O}$ possesses all properties of an indistinguishability obfuscator on $\mathcal{I}'$.
Firstly, we construct a subset $\mathcal{I}'$ from the family $\mathcal{I}$ that consists of all possible polynomial-size implementations for some program $P \in \mathcal{I}_\lambda$ of the original family. We formally construct $\mathcal{I}' \subseteq \mathcal{I}$ as follows:

1. Choose any $I \in \mathcal{I}$, add it to $\mathcal{I}'$ and define P as the program implemented by I.

2. Test every program $I' \in \mathcal{I}$ on whether it shows the same input-output behavior, meaning that its output $I'(x)$ is equivalent to the output $I(x)$ for all possible inputs x. The test condition can formally be written as

$$\forall x \in \{0,1\}^{\lambda(n)} \ : \ I'(x) \equiv I(x) \quad .$$

   Add all I' that pass the test to $\mathcal{I}'$.

$\mathcal{I}'$ thus is a valid family of polynomial-sized implementations for the same program P, as each $I \in \mathcal{I}'$ has identical input-output behavior. Furthermore, $\mathcal{I}'$ must be exhaustive, as every poly-sized implementation of P must have been included in $\mathcal{I}$. Otherwise, we could construct the following contradiction:
Given $\mathcal{I}'$, assume there is a polynomial-sized implementation $\tilde{I}$ of P not contained in $\mathcal{I}'$. This implies $\tilde{I}$ was not contained in $\mathcal{I}$, as it would otherwise have been added to $\mathcal{I}'$ during step 2 of the prior construction. As $\mathcal{I}$ defines the domain of $\mathcal{O}$ and its VBB indistinguishability is only guaranteed for implementations from the obfuscator's domain, this means that $\mathcal{O}$ can not obfuscate $\tilde{I}$.
We now choose any $I \in \mathcal{I}'$ with $I \neq \tilde{I}$. As both implementations are poly-sized, there exists a transformation T of at most $\mathsf{poly}(\lambda)$ steps from $\tilde{I}$ to I. To obfuscate $\tilde{I}$, $\mathcal{O}$ could then simply execute $T(\tilde{I})$ and obfuscate the resulting I afterwards, resulting in the mentioned contradiction.
We have yet to show that $\mathcal{O}$ qualifies as an indistinguishability obfuscation scheme of correctness ϕ_i on $\mathcal{I}'$, fulfilling the correctness, polynomial slowdown and indistinguishability properties of an $i\mathcal{O}$ scheme. We do so in the following steps by showing that the auditable obfuscation properties of $\mathcal{O}$ can be translated to the $i\mathcal{O}$ properties mentioned prior.

Correctness

We differentiate between the three correctness properties ϕ_1, ϕ_2 and ϕ_3 as specified in the definitions for obfuscation schemes in Section 2.3. We show that the definition for each property directly translates from auditable obfuscation to indistinguishability obfuscation.

ϕ_1 : ***(perfect correctness)***

As $\mathcal{O}$ suffices the definition of an auditable obfuscation scheme of correctness ϕ_1, we know that

$$\forall I \in \mathcal{I},\ \forall x \in \{0,1\}^{n(\lambda)}\ :\ \Pr_{\mathcal{O}}\left[\tilde{I}(x) = I(x) \mid \tilde{I} \leftarrow \mathcal{O}.\mathsf{Obf}(I)\right] = 1 \quad .$$

As $\mathcal{I}'$ is a subset of $\mathcal{I}$, it must also hold that

$$\forall I \in \mathcal{I}',\ \forall x \in \{0,1\}^{n(\lambda)}\ :\ \Pr_{\mathcal{O}}\left[\tilde{I}(x) = I(x) \mid \tilde{I} \leftarrow \mathcal{O}.\mathsf{Obf}(I)\right] = 1 \quad ,$$

which exactly is the ϕ_1 correctness of an indistinguishability obfuscation scheme defined over $\mathcal{I}'$.

ϕ_2 : ***(functionality preserving)***

As $\mathcal{O}$ fulfills the definition of an auditable obfuscation scheme of correctness ϕ_2, we know that

$$\forall I \in \mathcal{I}\ :\ \Pr_{\mathcal{O}}\left[\forall x \in \{0,1\}^{n(\lambda)} : \tilde{I}(x) = I(x) \mid \tilde{I} \leftarrow \mathcal{O}.\mathsf{Obf}(I)\right] > 1 - \mathsf{negl}(\lambda) \quad .$$

As $\mathcal{I}'$ is a subset of $\mathcal{I}$, it must also hold that

$$\forall I \in \mathcal{I}'\ :\ \Pr_{\mathcal{O}}\left[\forall x \in \{0,1\}^{n(\lambda)} : \tilde{I}(x) = I(x) \mid \tilde{I} \leftarrow \mathcal{O}.\mathsf{Obf}(I)\right] > 1 - \mathsf{negl}(\lambda) \quad ,$$

which precisely is the ϕ_2 correctness of an indistinguishability obfuscation scheme defined over $\mathcal{I}'$.

ϕ_3 : ***(weak functionality preserving)***

As $\mathcal{O}$ fulfills the definition of an auditable obfuscation scheme of correctness ϕ_3, we know that

$$\forall I \in \mathcal{I},\ \forall x \in \{0,1\}^{n(\lambda)}\ :\ \Pr_{\mathcal{O}}\left[\tilde{I}(x) = I(x) \mid \tilde{I} \leftarrow \mathcal{O}.\mathsf{Obf}(I)\right] > 1 - \mathsf{negl}(\lambda) \quad .$$

As $\mathcal{I}'$ is a subset of $\mathcal{I}$, it must also hold that

$$\forall I \in \mathcal{I}',\ \forall x \in \{0,1\}^{n(\lambda)}\ :\ \Pr_{\mathcal{O}}\left[\tilde{I}(x) = I(x) \mid \tilde{I} \leftarrow \mathcal{O}.\mathsf{Obf}(I)\right] > 1 - \mathsf{negl}(\lambda) \quad ,$$

which exactly is the ϕ_3 correctness of an indistinguishability obfuscation scheme defined over $\mathcal{I}'$.

The Obfuscator $\mathcal{O}$ of correctness ϕ_i thus possesses the correctness property ϕ_i of an indistinguishability obfuscation scheme in all cases.

Polynomial Slowdown
From the definition of an auditable obfuscation scheme, we know that, due to its own polynomial slowdown property, the running time of $\mathcal{O}.\mathsf{Obf}$ for an implementation $I \in \mathcal{I}$ must be bounded by a polynomial q with $q(|I|)$.
As $\mathcal{I}' \subseteq \mathcal{I}$, it thus also holds that for all $I \in \mathcal{I}'$, the runtime of $\mathcal{O}.\mathsf{Obf}$ must be bounded by a polynomial q with $q(|I|)$ such that for all relevant implementations, $\mathcal{O}$ suffices the polynomial slowdown property of an indistinguishability obfuscation scheme defined over $\mathcal{I}'$.

Indistinguishability
We assume $\mathcal{O}$ to be an auditable obfuscation scheme defined over $\mathcal{I}$. Obfuscator $\mathcal{O}$ thus fulfills its VBB indistinguishability property as given in Section 2.3, meaning

$$\forall I_0, I_1 \in \mathcal{I}, \forall \mathcal{A}\ \exists \mathcal{S}:$$
$$\left| |\Pr_{\mathcal{O}}[\mathcal{A}(\mathcal{O}.\mathsf{Obf}(I_0)) = 1] - \Pr_{\mathcal{O}}[\mathcal{A}(\mathcal{O}.\mathsf{Obf}(I_1)) = 1]| - |\Pr_{\mathcal{S}}[\mathcal{S}^{I_0}(1^\lambda) = 1] - \Pr_{\mathcal{S}}[\mathcal{S}^{I_1}(1^\lambda) = 1]| \right|$$
$$\leq \mathsf{negl}(\lambda) \quad .$$

As $\mathcal{I}' \subseteq \mathcal{I}$, we can deduce that

$$\forall I_0, I_1 \in \mathcal{I}', \forall \mathcal{A}\ \exists \mathcal{S}:$$
$$\left| |\Pr_{\mathcal{O}}[\mathcal{A}(\mathcal{O}.\mathsf{Obf}(I_0)) = 1] - \Pr_{\mathcal{O}}[\mathcal{A}(\mathcal{O}.\mathsf{Obf}(I_1)) = 1]| - |\Pr_{\mathcal{S}}[\mathcal{S}^{I_0}(1^\lambda) = 1] - \Pr_{\mathcal{S}}[\mathcal{S}^{I_1}(1^\lambda) = 1]| \right|$$
$$\leq \mathsf{negl}(\lambda) \quad .$$

Furthermore, as both I_0 and I_1 now implement the same program P defining $\mathcal{I}'$ we know that

$$\forall\, x \in \{0,1\}^{n(\lambda)} \ :\ I_0(x) \equiv I_1(x) \quad ,$$

from which we can further deduce that

$$\forall I \in \mathcal{I}' \ :\ \Pr_{\mathcal{S}}\left[\mathcal{S}^I(1^\lambda) = 1\right] = \frac{1}{|\mathcal{I}'|} \quad ,$$

as the input-output behavior is identical for all implementations. Simulators, by defini-

tion, only observe the input-output behavior via their assigned oracle. Based on these two facts, there can be no simulator achieving a better result than uniformly guessing which implementation was used, as such a simulator would imply differing input-output behavior being witnessed.
In the VBB indistinguishability term we fix the implementations before the simulators. We can then assume that any simulator in consideration is aware of this fixing and only ever decides between the choices I_0 and I_1. We can thus can restate our deduction with added quantifiers as

$$\forall I_0, I_1 \in \mathcal{I}', \forall \mathcal{S} \ : \ \Pr_{\mathcal{S}}\left[\mathcal{S}^I(1^\lambda) = 1 \middle| I \leftarrow \{I_0, I_1\}\right] = \frac{1}{|\{I_0, I_1\}|} = \frac{1}{2} \quad .$$

For any choice between two implementations from $\mathcal{I}'$, the probability of any simulator choosing correctly thus is $\frac{1}{2}$.
In the VBB indistinguishability, the simulator is only existentially quantified. As we quantified over all simulators in the equation above, we showed a strictly stronger property, allowing us to use it without adding constraints. Inserting this result into the VBB indistinguishability term defined over $\mathcal{I}'$ yields

$$\forall I_0, I_1 \in \mathcal{I}', \forall \mathcal{A}\, \exists \mathcal{S} \quad : \left| |\Pr_{\mathcal{O}}[\mathcal{A}(\mathcal{O}.\mathsf{Obf}(I_0)) = 1] - \Pr_{\mathcal{O}}[\mathcal{A}(\mathcal{O}.\mathsf{Obf}(I_1)) = 1]| - \left|\frac{1}{2} - \frac{1}{2}\right|\right|$$
$$\leq \mathsf{negl}(\lambda) \quad .$$

Simplifying results in

$$\forall I_0, I_1 \in \mathcal{I}', \forall \mathcal{A} \quad : \left|\Pr_{\mathcal{O}}[\mathcal{A}(\mathcal{O}.\mathsf{Obf}(I_0)) = 1] - \Pr_{\mathcal{O}}[\mathcal{A}(\mathcal{O}.\mathsf{Obf}(I_1)) = 1]\right| \leq \mathsf{negl}(\lambda) \quad .$$

As this term is equivalent to the indistinguishability property of an indistinguishability obfuscation scheme defined over $\mathcal{I}'$, the obfuscator $\mathcal{O}$ fulfills the indistinguishability property.

We have now proven that $\mathcal{O}$ possesses all necessary properties of an indistinguishability obfuscation scheme of correctness ϕ_i defined over $\mathcal{I}' = \{P\}$. As our choice of P was arbitrary, this must hold true for any $P \in \mathcal{I}$. □

From our proof it then follows that auditable obfuscation implies indistinguishability obfuscation. However, we have not yet established that the inherent subversion-resilience of $\mathcal{AO}$ is transferred. We investigate this in the following section.
Similarly, we have not yet discussed the inverse case of constructing any $\mathcal{AO}$ scheme from some $i\mathcal{O}$ scheme. While the general case with $|\mathcal{P}| > 1$ doesn't look promising, as indistin-

guishability obfuscation is defined over exactly one program class with no security guarantees when its domain is expanded, we can not inherently dismiss the case for $|\mathcal{P}| = 1$. Analyzing this special case, however, is not as straightforward as it may seem. As such, we do not investigate this case any further and discuss some of the difficulties in dealing with it in Appendix A.

5.2 Subversion-Resilient Indistinguishability Obfuscation

In the last section, we showed that an existing auditable obfuscation scheme on some $\mathcal{I}$ implies an existing indistinguishability obfuscation scheme for any program P as long as $P \in \mathcal{I}$. We have done so by proving that $\mathcal{AO}$ fulfills all properties of an $i\mathcal{O}$ scheme when run on the extracted set of all implementations for a program $P \in \mathcal{I}$.
As we did not modify or adapt our $\mathcal{AO}$ scheme beyond limiting its input realm, it seems trivially correct that the resulting $i\mathcal{O}$ scheme inherits any additional properties the $\mathcal{AO}$ scheme possessed. However, it might be that, in limiting the input realm, we removed a prerequisite for some of these additional properties. For any property inheritance beyond what was shown in the proof of Theorem 5.1, we should thus still rigorously prove, whether the property transfers correctly.
Most urgently, we are want to know whether we transferred the subversion-resilience of the $\mathcal{AO}$ scheme to the $i\mathcal{O}$ scheme. By definition, we know that the subversion-resilience of $\mathcal{AO}$ stems from its verifiability and soundness properties. For a proof by contradiction, let us now assume that we have not transferred the subversion-resilience in limiting the input realm, meaning that $\mathcal{AO}$ is not an auditable obfuscator on $\mathcal{I}'$. Using the following lemma, we shall disprove this assumption and show that, similarly to hardcore-bit assumptions [KL14], $\mathcal{AO}$ must be an auditable obfuscator on any (*non-singleton*) subset of $\mathcal{I}$. Note that the singleton case, meaning $|\mathcal{I}| = 1$, can inherently never be defined for $\mathcal{AO}$ and $i\mathcal{O}$, as both obfuscation schemes have their respective indistinguishability properties defined over at least two differing implementations in their implementation sets.

Lemma 5.2.
Let $\mathcal{O}$ be an auditable obfuscation scheme on $\mathcal{I}$. Then, for any $\mathcal{I}' \subseteq \mathcal{I}$ with $|\mathcal{I}'| > 1$, $\mathcal{O}$ also is an auditable obfuscation scheme on $\mathcal{I}'$.

Proof. Choose $\mathcal{I}'$ arbitrarily from the set of all subsets of $\mathcal{I}$ that are at least of size two.
Assume that $\mathcal{O}$ is not an auditable obfuscation scheme on $\mathcal{I}'$, meaning that either its correctness, polynomial slowdown, indistinguishability, verifiability or soundness property

must break when executing on $\mathcal{I}'$. However, by definition we know that an $\mathcal{AO}$ scheme's correctness, polynomial slowdown, indistinguishability, verifiability and soundness properties are all quantified over all implementations in $\mathcal{I}$. All properties must be inherently quantified over all implementations in $\mathcal{I}'$. The obfuscator $\mathcal{O}$ must thus also fulfill all properties of an auditable obfuscation scheme on $\mathcal{I}'$, resulting in a contradiction. □

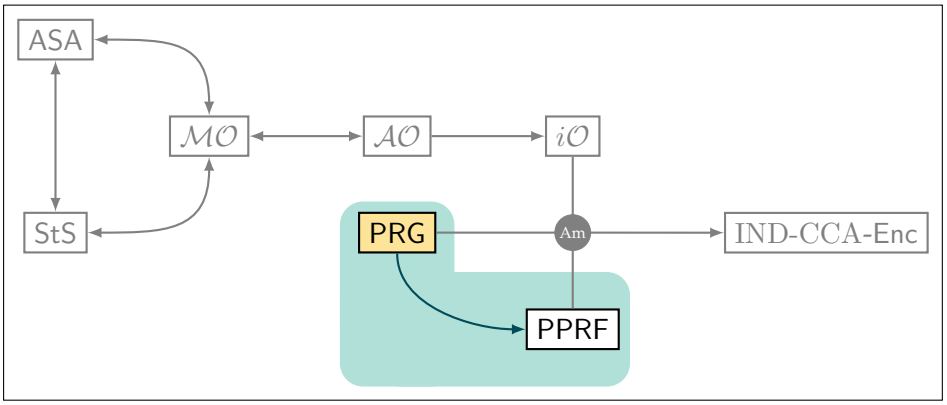
Having shown that $\mathcal{AO}$ must be an auditable obfuscator on any (*non-singleton*) subset of $\mathcal{I}$, we can now achieve our original goal of proving that, using the construction from Theorem 5.1, we transferred the subversion-resilience of the $\mathcal{AO}$ scheme to the constructed $i\mathcal{O}$ scheme.

Corollary 5.3.
The $i\mathcal{O}$ scheme described by Theorem 5.1 is subversion-resilient.

Proof. Let $\mathcal{O}$ be the $i\mathcal{O}$ scheme described by Theorem 5.1.
The obfuscator $\mathcal{O}$ thus is an auditable obfuscation scheme on $\mathcal{I}$ working on $\mathcal{I}'$ with $\mathcal{I}' \subseteq \mathcal{I}$. By construction, $\mathcal{I}'$ consists of all polynomial-sized implementations of some program $P \in \mathcal{I}$. Trivially, $|\mathcal{I}'|$ must thus be of at least size 2, as for each polynomial-sized implementation $I \in \mathcal{I}'$ we can find a second polynomial-sized implementation $\tilde{I} \in \mathcal{I}'$ that simply uses an additional NOP operation before its execution.
Following from Lemma 5.2, $\mathcal{O}$ must then be an auditable obfuscation scheme and, due to Theorem 4.3, subversion-resilient. □

With this, the goal of this chapter has been achieved. We were able to relate auditable obfuscation to indistinguishability obfuscation, demonstrating that a valid $\mathcal{AO}$ scheme can be leveraged to construct a valid $i\mathcal{O}$ scheme. Furthermore, we were able to show that the provided construction transfers the auditability of $\mathcal{AO}$ to the resulting $i\mathcal{O}$ scheme, thereby making it subversion-resilient.
In the next chapter, we shall discuss pseudorandom generators and pseudorandom functions as well as their relation to subversion-resilience. Afterwards, we will combine the results of this chapter and the next in Chapter 7 to achieve subversion-resilient CCA-secure encryption.

6 Constructing Subversion-Resilient Puncturable Pseudorandom Functions



In the last chapter, we discussed the relation between auditable obfuscation and indistinguishability obfuscation and were able to show that we could leverage $\mathcal{AO}$ to achieve subversion-resilient $i\mathcal{O}$. With this, we not only provided a first use case example for our results from Chapter 4, but the first subversion-resilient building block for our work in Chapter 7 as well.

In Chapter 7, we will use the Sahai and Waters construction for CCA-secure encryption from $i\mathcal{O}$ [SW14] to achieve subversion-resilient CCA-secure encryption. However, besides $i\mathcal{O}$, the construction method leverages pseudorandom generators (PRG) and puncturable pseudorandom functions (PPRF) to achieve its security. As discussed in the technical overview of Chapter 3, we need require of these primitives to be subversion-resilient as well.

In this chapter, we will look at both the PRG and PPRF primitive. For PRGs, we will show that any construction following the standard security game as depicted in Figure 2.7 inherently is subversion-resilient. On the other hand, we will show that PPRFs generally are not subversion-resilient, even if their underlying PRF is, by demonstrating that adversarial puncturing can be levied as a steganographic channel.

While we do show that subversion-resilient arbitrary-punctured PRFs are not achievable, we conclude this chapter by showing that subversion-resilient *randomly*-punctured PRFs do exist. As proof, we first build a subversion-resilient PRF using the Goldreich-Goldwasser-Micali ([GGM86]) construction for PRFs from PRGs and then prove that, in

fixed (*pseudo-*) randomly puncturing it, we do not break subversion-resilience.

6.1 Pseudorandom Generators are Subversion-Resilient

In most of modern computer science, cryptography in particular, randomness is a necessity for achieving good results [KL14]. However, in real-world settings, we are unable to produce true randomness, thus having to rely on something that comes as close to it as possible. Algorithms producing such *pseudorandomness* are called (*cryptographic*) pseudorandom generators (PRG).
In a cryptographic context, a PRG may either be a PPT algorithm that needs to be indistinguishable from randomness, even if started with a worst-case adversarial seed as input, or a deterministic polytime algorithm indistinguishable on any uniformly chosen input. Yao's Principle then guarantees us that both settings are equivalent [Wig19, AB09]. An informal statement of Yao's Principle can be found in Theorem 2.14.
When comparing the two possible settings with regards to any cryptographic analysis, the second setting seems easier to work with. As such, we will be using it throughout this chapter. A more formal definition of PRGs as deterministic algorithms as well as the associated cryptographic security game can be found in Section 2.4.
The construction of subversion-resilient IND-CCA-secure encryption in Chapter 7 heavily relies on the use of a subversion-resilient PRG. This section tackles the existence of such PRGs by proving the following Theorem 6.1, which states that any PRG fulfilling the security game as depicted in Figure 2.7 inherently is subversion-resilient. We utilize a similar argument as [BBD+23] in their proof of weak-PRFs being inherently subversion-resilient:
The adversary in both security games of weak-PRFs and PRGs does not provide any input to the executing algorithm before having to answer in their respective real-or-random game. This allows one to *fix* the input distribution and thus apply Lemma 2.13 from [RTYZ16] in the analysis. As a result, for any adversary with some advantage, there is a watchdog of equal strength protecting against them, meaning that any working subversion must also be detectable.

Theorem 6.1.
If G is a pseudorandom generator, then the trivial specification $\widehat{G} = G$ is a pseudorandom generator under subversion.

Proof. Assume G is a PRG. As such, G is a deterministic PPTM over input realm $\{0,1\}^{n(\lambda)}$. In the standard PRG game (*see Figure 2.7*), the distribution X_λ^G over the possible seeds

in $\{0,1\}^{n(\lambda)}$ is not known to the adversary $\mathcal{A}$ and could thus be arbitrary. In the trivial implementation, however, we assume the input space to be uniformly distributed when sampling s, such that we can publicly fix the input distribution X_λ^G to be uniform. Fixing and publishing of the input distribution provides the adversary $\mathcal{A}$ with additional information, meaning it can only strengthen $\mathcal{A}$.
By fixing X_λ^G, G becomes a deterministic PPTM working on a public input distribution. As such, we can apply Lemma 2.13 to its implementations. Let $\tilde{G}$ be a subverted implementation of G and $\widehat{G}$ its specification. Then let $\mathsf{Neq}_\lambda \subseteq \{0,1\}^{\lambda(n)}$ be the set of inputs at which $\tilde{G}$ deviates from $\widehat{G}$, yielding

$$\mathsf{Neq}_\lambda = \left\{ s \in \{0,1\}^{n(\lambda)} \mid \tilde{G}(s) \neq \widehat{G}(s) \right\} \quad .$$

Through Lemma 2.13, we now know that there exists a PPT offline watchdog with detection rate

$$\delta = \Pr\left[\tilde{G}(s) \neq \widehat{G}(s) \mid s \leftarrow\!\$\; X_\lambda^G\right] = \frac{|\mathsf{Neq}_\lambda|}{\left|\{0,1\}^{n(\lambda)}\right|}$$

for the subverted implementation $\tilde{G}$.
The goal of the adversary $\mathcal{A}$ is to exploit a given subversion in $\tilde{G}$ without being detected, meaning $\mathcal{A}$ plays against the subversion-resilience game. Any adversary in the subversion-resilience game wins if and only if no corresponding watchdog with a non-negligible detection rate exists. As such, for $\mathcal{A}$ to exploit the PRG implementation $\tilde{G}$ while avoiding detection, they must fulfill

$$\frac{|\mathsf{Neq}_\lambda|}{\left|\{0,1\}^{n(\lambda)}\right|} \leq \mathsf{negl}(\lambda) \quad .$$

Similarly, $\mathcal{A}$ can only win and subvert the PRG if and only if they cause any of the inputs to deviate from their specification. Let this be the event hit. For any polynomial-time adversary $\mathcal{A}$, the resulting success probability is thus capped by

$$\mathsf{Adv}^{\mathrm{sub-prg}}_{\mathcal{A},\mathsf{PRG}}(\lambda) = \#\mathsf{tries} \cdot \Pr[\mathsf{hit}] \leq \mathsf{poly}(\lambda) \cdot \frac{|\mathsf{Neq}_\lambda|}{\left|\{0,1\}^{n(\lambda)}\right|} \quad .$$

Combining both requirements now yields that

$$\mathsf{Adv}^{\mathrm{sub-prg}}_{\mathcal{A},\mathsf{PRG}}(\lambda) \leq \mathsf{poly}(\lambda) \cdot \frac{|\mathsf{Neq}_\lambda|}{\left|\{0,1\}^{n(\lambda)}\right|} \leq \mathsf{poly}(\lambda) \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda) \quad ,$$

meaning that there only is a negligible chance of $\mathcal{A}$ successfully exploiting a subversion in a given implementation $\tilde{G}$ of the specification $\widehat{G} = G$. □

We showed that any PRG fulfilling its trivial specification of winning the security game in Figure 2.7 with overwhelming probability is subversion-resilient. Thereby, we have successfully demonstrated that we can subversion-resiliently create two of the three primitives necessary for our construction.
The only primitive not addressed thus far are puncturable pseudorandom functions. As PPRFs are a modification to standard PRFs, we need to construct a subversion-resilient PRF as an intermediary step. While some constructions for subversion-resilient PRFs [BBD+23] already exist, PPRFs are generally instantiated using the Goldreich-Goldwasser-Micali construction, as the puncturing is well-defined in its structure [BGI14, BW13, KPTZ13, SW14, HSW14]. Relying on known construction methods, we will use the GGM construction as the base PRF in this thesis.
To our knowledge, there have been no publicized attempts to analyze the GGM construction with regards to subversion-resilience. We thus perform the analysis ourselves. In the next section, we will use the results of this section to prove that the resulting PRF of the GGM construction is subversion-resilient, given that its underlying PRG is subversion-resilient as well.

6.2 Constructing a Subversion-Resilient PRF

Before analyzing PPRFs with regards to subversion-resilience, we first need to construct a subversion-resilient PRF as their basis. As discussed in the last section, we use the Goldreich-Goldwasser-Micali construction as our underlying PRF, as its puncturing is well-understood.
We use this section to give a short introduction on the GGM construction, provide an intuition on the resulting tree structure, and prove that the GGM construction is subversion-resilient, given a subversion-resilient PRG is used.

6.2.1 The Goldreich-Goldwasser-Micali (GGM) Construction

The Goldreich-Goldwasser-Micali (GGM) construction was first established by Goldreich, Goldwasser and Micali in [GGM86]. The construction was used as a first example that fulfilled their formal definition of a *secure* pseudorandom function. As such, it is one of the most commonly used constructions. However, as notations and nomenclature have changed in the last few decades, instead of matching the original paper, we will match our notation more closely to the write-up of the construction provided in [KL14].
As the general structure of GGM is relevant to our subversion analysis, we will provide a quick overview of the construction and its functionality. For now, we will denote the resulting PRF of the GGM construction with F.

The general idea behind the construction of F is as follows. At first, the construction is provided some depth $\ell \in \mathbb{N}$ and a pseudorandom generator G. The depth ℓ must be fixed at construction time in $\mathsf{poly}(\lambda)$, while G must be keyed, meaning inputs advance the internal stepping of Gs seed value. Further, G must produce outputs that are double the length of the inputs, meaning G outputs values of length $2n(\lambda)$. Generator G can thus be described with the following signature

$$G : \{0,1\}^{n(\lambda)} \to \{0,1\}^{2n(\lambda)} \quad .$$

Both of these parameters are used to construct a binary tree, also called the GGM-tree, of depth ℓ, where in each internal node an instance of G is fixed. It must be noted that this tree is only created implicitly at this point. For execution, each instance of G is only fixed at runtime, meaning non-executed instances of G need not be stored. The leaves of the resulting binary tree are used as the outputs of F.
Executing F now works as follows. First, an input x of length ℓ is provided, together with a key k of length $n(\lambda)$. Then, the instantiated G at the root of the constructed GGM-tree is evaluated on k and produces an output $y = k_0^{(\varepsilon)} \| k_1^{(\varepsilon)}$. As the next step, depending on the most significant bit of x, the next internal node is chosen for execution. If x was a 0, then the GGM-tree is traversed to the left child of root with $k_0^{(\varepsilon)}$ as its key $k^{(0)}$, otherwise the GGM-tree is traversed to the right child of root with $k_1^{(\varepsilon)}$ as its key $k^{(1)}$.
From here on out, the execution continues analogously for all levels of the GGM-tree, in each layer i checking the relevant bit $x[i]$ from the input instead of the most significant bit. After completely traversing the GGM-tree, the execution reaches one of the trees leaves. This leaf contains the output from the last layer, meaning $k^{(x)}$, which, as mentioned before, is used as the output of F.
Generally, we write $F^{(\ell)}$ for any F with a fixed, known depth ℓ. The signature for $F^{(\ell)}$ then is

$$F^{(\ell)} : \underbrace{\{0,1\}^{n(\lambda)}}_{\mathcal{K}} \times \underbrace{\{0,1\}^{\ell}}_{\mathcal{D}} \to \underbrace{\{0,1\}^{n(\lambda)}}_{\mathcal{R}}$$

where $\mathcal{K}$ is the domain of all valid keys, $\mathcal{D}$ the domain of all valid inputs and $\mathcal{R}$ the output domain. We provide an example construction for a PRF using a GGM-tree in Figure 6.1.

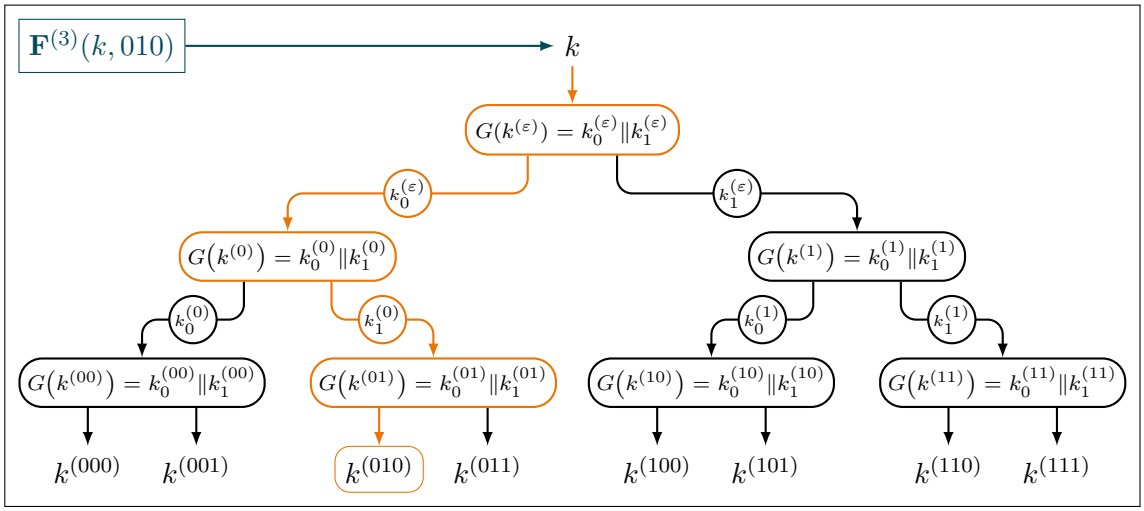


Figure 6.1: Visualization of the GGM-tree with depth $\ell = 3$. The colored path displays the evaluation path for the call to the construction with input $x = 010$ and key k. The function returns the value $k^{(010)}$.

The last thing to look at regarding the GGM construction are its pseudorandomness guarantees. The pseudorandomness of the construction is formalized with the following Theorem 6.2 due to [GGM86].

Theorem 6.2 ([GGM86]).
If $G : \{0,1\}^{n(\lambda)} \to \{0,1\}^{2n(\lambda)}$ is a pseudorandom generator, then the function $F^{(\ell)}$ following the prior outlined construction is pseudorandom.

The proof of the theorem relies on the pseudorandomness property of the underlying pseudorandom generator. Each output at any layer is the output of a PRG on a key of its input domain, meaning these outputs are not distinguishable from randomness. As the leaves only contain outputs from the PRGs as well, it would then conclude that these must also be pseudorandom. Otherwise, an attacker on the PRG scheme could be constructed, that works by simulating the attacker against the GGM construction on $F^{(1)}$. A thorough write-up on the proof of Theorem 6.2 can be found in [KL14].
With a firm grasp on the GGM construction we can now start with our subversion analysis. In the next section, we will show that we can leverage a subversion-resilient PRG in combination with the treelike structure of GGM to prove the construction subversion-resilient.

6.2.2 GGM constructed PRFs are Subversion-Resilient

We want to prove that the GGM construction results in a subversion-resilient PRF if the underlying PRG is subversion-resilient. We formalize this statement in the following Theorem 6.3.

Theorem 6.3.
Let $\widehat{F^{(\ell)}}$ be a PRF *following the GGM-construction of depth ℓ using* PRG *G.*
If G is a pseudorandom generator under subversion, then for each $\ell \in \mathbb{N}$, $\widehat{F^{(\ell)}}$ is pseudorandom under subversion.

Proof. We will follow a proof by induction. First we will show that $\widehat{F^{(1)}}$ is pseudorandom under subversion. Afterwards, we will extrapolate the result to $\widehat{F^{(\ell)}}$ for each $\ell \in \mathbb{N}$.
Pseudorandom function $\widehat{F^{(1)}}$ is given the inputs k, x with $k \in \{0,1\}^{n(\lambda)}$ and $x \in \{0,1\}$. The output is derived by calling the pseudorandom generator G on k and then splicing the output of G into two strings of length $n(\lambda)$. If x is 0, then the first of the two strings, $k_0^{(\varepsilon)}$, is returned, else the second, $k_1^{(\varepsilon)}$.
As the output of $\widehat{F^{(1)}}$ is simply a truncated version of the output of G it must hold that any subversion in the output of $\widehat{F^{(1)}}$ was already present in the output of G. Otherwise, we could create an attacker on G by simulating an attacker on $\widehat{F^{(1)}}$. Thus, any bound on achievable subversions on G must also hold for $\widehat{F^{(1)}}$.
As shown in Theorem 6.1, we can always construct a watchdog for G that simply samples random keys as input, given the input distribution is public. As a pseudorandom function must work for any key from its input distribution, we can leverage this and assume that k is from a uniform distribution, meaning $k \leftarrow\!\$\ \{0,1\}^{n(\lambda)}$.
Now let Neq_λ be the set of inputs on which $\tilde{G}$ deviates from its specification. We then know that the probability $p = \frac{|\mathsf{Neq}_\lambda|}{|\{0,1\}^{n(\lambda)}|}$ of an attacker hitting this set is negligible. We can use this to build a watchdog for $\widehat{F^{(1)}}$ that guarantees the correctness with overwhelming probability.
Further, let $\ell \in \mathbb{N}$ with $\ell > 1$ be the height of the GGM construction. The PRF $\widehat{F^{(\ell)}}$ is thus called with inputs k, x where $k \in \{0,1\}^{n(\lambda)}$ and $x \in \{0,1\}^{\ell}$. The first layer of the GGM construction works analogous to $\widehat{F^{(1)}}$ and each layer afterwards works by calling G with a partial result from the layer before. Which partial result is used thereby depends on x, specifically:

$$\widehat{F^{(i+1)}}(k,x) = \begin{cases} G\left(k_0^{(x[0:i])}\right) & \text{, if } x[i+1] = 0 \\ G\left(k_1^{(x[0:i])}\right) & \text{, if } x[i+1] = 1 \end{cases}$$

Each layer thus only depends on the output from the layer beforehand.
A polytime adversary may now test polynomial combinations of k and x for a given implementation of $\widehat{F^{(\ell)}}$ to try and find a subversion. We assume that if any of the intermediary evaluations of G in $\widehat{F^{(\ell)}}$ results in a subverted output of G that this subversion is persistent from that point forward. If the subversion would not be persistent, then an adversary would not see it, as we do not output intermediary results. Let the event that a subversion was achieved in layer i now be denoted by hit_i, then the statement can be written as

$$\Pr\left[\mathrm{hit}_i \middle|\, \mathrm{hit}_{i-1}\right] = 1 \quad .$$

As such, for any output in $\widehat{F^{(\ell)}}$ the output is subverted, if and only if, the input of the last layer is in Neq_λ or the output of the prior layer was already subverted. For any single evaluation on fixed x, k the probability of the output of $\widehat{F^{(\ell)}}$ being subverted is thus

$$\Pr\left[\mathrm{hit}_\ell\right] = \Pr\left[\mathrm{hit}_\ell \;\middle|\; \overline{\mathrm{hit}_{\ell-1}}\right] \cdot \Pr\left[\overline{\mathrm{hit}_{\ell-1}}\right] + \Pr\left[\mathrm{hit}_\ell \;\middle|\; \mathrm{hit}_{\ell-1}\right] \cdot \Pr\left[\mathrm{hit}_{\ell-1}\right] .$$

When $F^{(\ell-1)}$ was not subverted, the last layer of $F^{(\ell)}$ simply acts as an execution of the underlying PRG G on a pseudorandom input. This setting is equivalen to that of Theorem 6.1, thus we know that $\Pr\left[\mathrm{hit}_\ell \;\middle|\; \overline{\mathrm{hit}_{\ell-1}}\right] = \frac{|\mathsf{Neq}_\lambda|}{|\{0,1\}^{n(\lambda)}|}$, which is negligible. As we also assumed any subversion to be persistent, we can then write

$$\begin{aligned}
\Pr\left[\mathrm{hit}_\ell\right] &= p \cdot \Pr\left[\overline{\mathrm{hit}_{\ell-1}}\right] + 1 \cdot \Pr\left[\mathrm{hit}_{\ell-1}\right] \\
&= \mathsf{negl}(\lambda) \cdot \Pr\left[\overline{\mathrm{hit}_{\ell-1}}\right] + \Pr\left[\mathrm{hit}_{\ell-1}\right] \\
&= \mathsf{negl}(\lambda) + \Pr\left[\mathrm{hit}_{\ell-1}\right] \quad .
\end{aligned}$$

We can analyze the probability of $\widehat{F^{(\ell-1)}}$ and all following intermediary steps analogously until $\widehat{F^{(1)}}$ and substitute them into the term. Further, for $\widehat{F^{(1)}}$ we have already shown the probability of a subversion to be negligible. Thus we get that

$$\Pr\left[\mathrm{hit}_\ell\right] = \Pr\left[\mathrm{hit}_1\right] + \sum_{i=2}^{\ell} \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda) + \sum_{i=2}^{\ell} \mathsf{negl}(\lambda) = \ell \cdot \mathsf{negl}(\lambda) \quad .$$

Depth ℓ is by assumption fixed at construction time and at most in $\mathsf{poly}(\lambda)$, meaning that

$$\Pr\left[\mathrm{hit}_\ell\right] \leq \mathsf{poly}(\lambda) \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda) \quad .$$

Any adversary may now test polynomial many combinations of k and x. We will model this as the adversary being allowed to choose polynomial many k and for each k polyno-

mial many x. From this we get that

$$\mathsf{Adv}^{\mathrm{sub-ggm}}_{\mathcal{A},\mathsf{PRF}}(\lambda) = \sum_{k}\sum_{x} \Pr[\mathrm{hit}_\ell] \leq \mathsf{poly}(\lambda)\cdot\mathsf{poly}(\lambda)\cdot\mathsf{negl}(\lambda) = \mathsf{negl}(\lambda) \quad .$$

The GGM construction for $\widehat{F^{(\ell)}}$ is thus pseudorandom under subversion. □

We were able to show that the GGM construction for PRFs from PRGs is subversion-resilient, given a subversion-resilient PRG is used. With this, we now have all necessary components to tackle puncturable PRFs.

6.3 Subversion-Resilient Puncturable Pseudorandom Functions

In the last section, we showed that the GGM construction provides a subversion-resilient PRF, given a subversion-resilient PRG is used. As stated beforehand, most PPRFs are adapted from PRFs, built using GGM. This is due to the inherent tree-like structure encountered within them, the GGM-tree. As each differing execution of a GGM-PRF corresponds to a path through the GGM-tree, we can create a PPRF from it by simply *truncating* subtrees containing the punctured leaves, meaning we do not allow these paths to be followed to their conclusion [BGI14, BW13, KPTZ13]. So long as we do not leak any internal states from a common ancestor of any of our truncations we then fulfill the formal definition of a PPRF. To visualize the puncturing process, we provide an example construction in Figure 6.2.

We now want to analyze PPRFs under subversion, meaning an adversary tries to maliciously modify the implementation in some way. As we know from [BL17, Ber18], the subversion setting is equivalent to the steganographic setting on a certain type of channel. In the steganographic setting, the adversaries goal is to communicate some kind of information to an external party without anyone noticing this communication. For our analysis of PPRF the second setting seems more favorable to work with.

As a PPRF is always a modification to an underlying PRF, and we already know that subversion-resilient PRFs do exist (see [BBD+23] and Section 6.2), we can simplify our analysis by using such a subversion-resilient PRF as basis. We then only need to show, that none of the modifications made when translating the construction to a PPRF created new attack vectors.

Unfortunately, we can show that any adversary-provided input for puncturing can be used to create a steganographic channel. We prove this with the following Lemma 6.4.

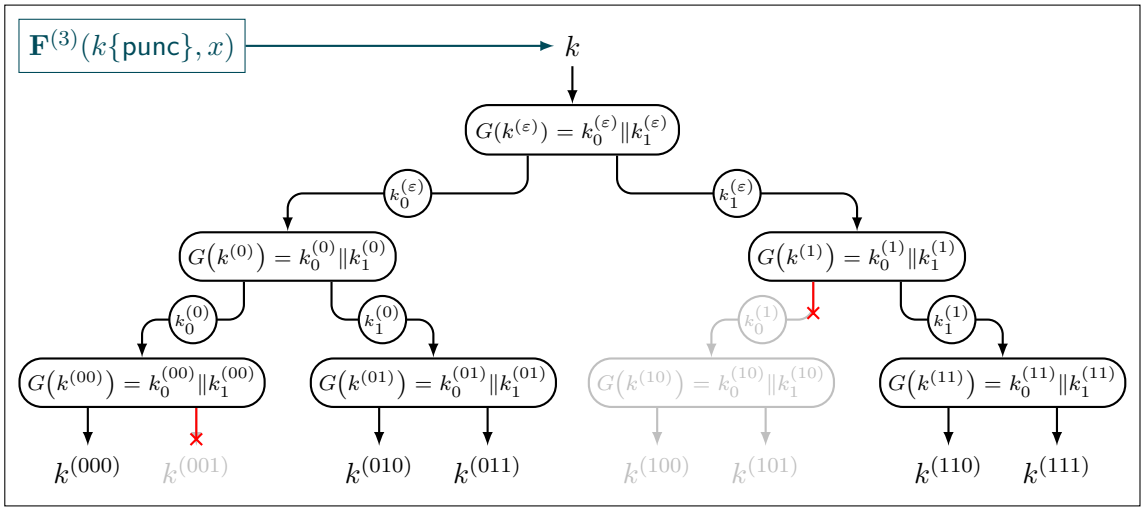


Figure 6.2: Visualization of a GGM-tree with depth $\ell = 3$ that has been punctured at the positions $\mathsf{punc} = \{001, 100, 101\}$. Grayed out parts of the GGM-tree depict the puncturing, meaning that these paths can not be evaluated using $\mathsf{k}\{\mathsf{punc}\}$. If $F^{(3)}$ is called with $x \in \mathsf{punc}$, instead of the correct outputs, we can assume that the function either outputs $\perp$ or silently returns. For $x \notin \mathsf{punc}$ the function outputs the non-punctured values, meaning $F^3(\mathsf{k}, x)$.

Lemma 6.4.
Let F be a punctured pseudorandom function with user-controlled puncture-positions punc *based on a* PRF *that is pseudorandom under subversion.*
Then, F is not subversion-resilient.

Proof. Let F be a PPRF based on a subversion-resilient PRF and let $\mathcal{A}$ be the adversarial user. We show constructively that F contains a steganographic channel, despite of the subversion-resilience of PRF.
When F is instantiated, $\mathcal{A}$ is queried on which positions of F they want to puncture. Let $[y_0, y_1, \ldots, y_{2^\ell}]$ be all possible outputs of F, indexed by their associated input. If $\mathcal{A}$ wants to send a 0, they choose an arbitrary y_i with $i \equiv 0 \bmod 2$ and puncture it. Otherwise, if $\mathcal{A}$ wants to send a 1, they choose an arbitrary y_i with $i \equiv 1 \bmod 2$ and puncture it.
As the positions of puncturing must be encoded in the puncturing key $\mathsf{k}\{\mathsf{punc}\}$, where punc is the set of all punctured positions, the encoded message can be extracted by simply checking the above mentioned condition.
In the chosen hiddentext game for StS, as described by Figure 2.1, a warden W is allowed to specify the message sent by $\mathcal{A}$. Trivially, if $\mathcal{A}$ does not modify the message before em-

bedding it, any warden has a non-negligible probability of detecting this encoding. They achieve this by simply checking if there exists a common predicate between the puncturing when their chosen message is a 1, or vice versa a 0.
Thus, let us now assume that $\mathcal{A}$ has access to a IND$-CPA-secure encryption function Enc. $\mathcal{A}$ can now call Enc on the warden-controlled message msg and puncture depending on the bit(s) of this encryption. The extraction works similarly identical, but must decode the received cipher using the associated Dec function. See Figure 6.3 for a pseudocode example of these two functions. As the encryption scheme is IND$-CPA-secure, the output of it

embedPunctureKey (*hidden message* msg, *embedding key* ak, PPRF $\widehat{F^{(\ell)}}$)

1 : punc $\leftarrow$ Enc(msg, ak)
2 : k $\leftarrow \widehat{F^{(\ell)}}$.Gen($\cdot$)
3 : k{punc} $\leftarrow \widehat{F^{(\ell)}}$.Punc(k, punc)
4 : **return** k{punc}

extractPunctureEmbed (*punctured key* k{punc}, *embedding key* ak)

1 : punc $\leftarrow$ Extr(k{punc})
2 : msg* $\leftarrow$ Dec(punc, ak)
3 : **return** msg*

Figure 6.3: Malicious puncturing for a PPRF using the subversion-resilient PRF construction from Section 6.2. The construction uses GGM of depth ℓ and allows for a user-specified puncturing of one element at its lowest level. An adversary can now encode a 1-bit message via the point of puncture. As the punctured key by necessity needs to contain the information, which position was punctured, any party interacting with the PPRF and its punctured key can thus gain access to the encoded message.

looks indistinguishable from randomness, meaning the resulting positions of puncturing look random as well. Now, any warden that could still meaningfully distinguish between these puncturings and random positions, must thus be able to break the encryption. The best advantage of any $\mathcal{A}_{\mathsf{Enc}}$ must then be an upper bound on the advantage of W.
As such, it follows that

$$\mathsf{Adv}^{\mathrm{ss-cha}}_{\mathsf{W},\mathsf{StS},\mathcal{C}_{F_{\mathsf{punc}}}}(\lambda) \leq \mathsf{InSec}^{\mathsf{enc-cpa}}_{\mathsf{Enc}}(\lambda) \quad .$$

However, as Enc is a secure encryption function, it further follows that

$$\mathsf{InSec}^{\mathsf{enc-cpa}}_{\mathsf{Enc}}(\lambda) \leq \mathsf{negl}(\lambda) \quad ,$$

meaning that any warden on the constructed steganographic channel only has at most a negligible advantage. This implies the existence of a subversion against F. □

We were able to create a steganographic channel using a PPRF from a subversion-resilient PRF, as the steganographic channel did not depend on the outputs, but rather on which positions were used in the puncturing. For the sake of conciseness, we limited ourselves to a rather simple encoding. However, more complex constructions are able to send up to 2^ℓ possible messages in one encoding. We use Appendix B to cover some of these more efficient methods an adversary could incorporate.
Even with the simple encoding and limited to one single puncture position we were still able to describe a steganographic channel. From this, we can conclude that any PPRF with user-controlled puncturing must be vulnerable to subversions.
However, this does not mean that all PPRFs can be subverted. Similar to other primitives, we might be able to construct a subversion-resilient variant by removing the input from users and substituting it with randomness. One such possible attempt would be to force the positions for puncturing to be drawn by a subversion-resilient PRG. Thankfully, the Sahai and Waters construction already calls for random puncturing in their proof [SW14], meaning this additional constraint would have been necessary even if arbitrary puncturing could work subversion-resiliently.
Sadly, as we will see, simply drawing the positions by random is not enough. The adversary still has some leverage left, namely if they use any randomness or redraw it. This technique is commonly known as rejection sampling [And96, Cac04, Ber18] and is used quite often in the design of steganographic systems. In the following lemma, we show how we can leverage it against our modification to construct a steganographic channel.

Lemma 6.5.
Let F be a punctured pseudorandom function based on a subversion-resilient PRF *that has its puncture-positions* punc *chosen by a* PRG *that is pseudorandom under subversion.*
Then, F is not subversion-resilient if an adversary $\mathcal{A}$ can use rejection sampling.

Proof. Let F be a PPRF based on a subversion-resilient PRF that uses a subversion-resilient PRG during initialization of its punctured positions. Further, let $\mathcal{A}$ be the adversarial user. We show constructively that F contains a steganographic channel, despite of the subversion-resilience of PRG and PRF.
When F is instantiated, the PRG is queried and produces some puncturing set punc. Before punc is used in the puncturing, $\mathcal{A}$ now intercepts punc and analyzes it. For this, $\mathcal{A}$

interprets the positions in punc as describing a repetition code $\mathrm{mod}2$. More precisely, for each position i in punc $\mathcal{A}$ tests if $i \equiv 0 \bmod 2$ or $i \equiv 1 \bmod 2$ and sums the results. If more than half of the positions evaluate to a 0, then $\mathcal{A}$ interprets punc as a repetition codeword for a 0. Vice versa, if more than half of the positions evaluate to a 1, then $\mathcal{A}$ interprets punc as a 1. If neither is the case, then punc is interpreted as an invalid codeword.
For a given, uniformly random position set punc of size m, we have the following probabilities of occurrence

$$\Pr_{\mathsf{punc}}[\mathcal{A} \text{ interprets punc as a } 0] = \frac{\sum_{i=0}^{\lceil \frac{m}{2}-1 \rceil} \binom{m}{i}}{\sum_{i=0}^{m} \binom{m}{i}} = \frac{2^{m-1} - \frac{\binom{m}{m/2}}{2}}{2^m} = p$$

$$\Pr_{\mathsf{punc}}[\mathcal{A} \text{ interprets punc as a } 1] = \frac{\sum_{i=\lfloor \frac{m}{2}+1 \rfloor}^{m} \binom{m}{i}}{\sum_{i=0}^{m} \binom{m}{i}} = \frac{2^{m-1} - \frac{\binom{m}{m/2}}{2}}{2^m} = p$$

$$\Pr_{\mathsf{punc}}[\mathcal{A} \text{ interprets punc as invalid}] = \frac{\binom{m}{m/2}}{\sum_{i=0}^{m} \binom{m}{i}} = \frac{\binom{m}{m/2}}{2^m} \quad ,$$

where the randomness is taken over the choices of elements in punc. We note, that for odd values of m the term $\binom{m}{m/2}$ is not well-defined, as $\frac{m}{2}$ is not a whole number. In abuse of notation, we define the output in this case as 0.
After the analysis of punc, $\mathcal{A}$ compares the currently encoded bit with the message they want to send. With above stated probability p the set already encodes the message. In this case, $\mathcal{A}$ simply continues the instantiation of F.
However, with inverse probability $1 - p$, the set does not encode the correct message. In this case, punc is dropped and a new random set is queried. Adversary $\mathcal{A}$ only fails if they can not query a *good* set in poly-time, meaning each of their $\mathsf{poly}(\lambda)$ queries to the subversion-resilient PRG must fail. As each draw is independent, we can describe the probability of this case with the following term:

$$\begin{aligned}
\Pr_{\mathsf{punc}}[\mathcal{A} \text{ fails}] &= \prod_{\mathsf{poly}(\lambda)} \Pr_{\mathsf{punc}}[\mathsf{punc}_i \text{ does not work for } \mathcal{A}] \\
&= \left(\frac{2^{m-1} + \frac{\binom{m}{m/2}}{2}}{2^m} \right)^{\mathsf{poly}(\lambda)} \\
&\approx \left(\frac{1}{2} \right)^{\mathsf{poly}(\lambda)} = \frac{1}{2^{\mathsf{poly}(\lambda)}} = \mathsf{negl}(\lambda) \quad .
\end{aligned}$$

We can now continue the analysis analogous to the proof of Lemma 6.4 by allowing $\mathcal{A}$ to use a secure encryption scheme. From this, we get that there exists a steganographic channel where any warden W only has at most a negligible advantage when trying to detect it. This implies a subversion against F. □

We have shown that, even if an adversary is only allowed to use or refuse any choice of puncture positions, they are still able to construct a stegosystem StS. While there may be ways to fix the above construction by additional puncturing after punc is provided, it would take exponential work to defend against all possible adversaries.
As we generally only consider *computable* defenses, meaning running in at most poly-time, we must try a different strategy. One such possible strategy is, instead of fixing possible *bad* choices for punc, to remove the adversaries means of obtaining them.
In our first two attempts, the adversary was either able to directly choose the puncturing or indirectly choose it via rejection sampling. Preventing both would mean that we force the adversary to comply with any choice of randomness we provide, or that any choice they make gets overwritten by some further rerandomizing before usage. More concise, we revoke the adversaries sampling rights. An example of this modified security game using rerandomization can be seen in Figure 6.4.
We note, that this prerequisite could also be stated as a distribution requirement. Namely, we would need not only any individual choice of a set punc to follow a subversion-resilient distribution, but the distribution over all sets punc and their context itself to be subversion-resilient as well.
For this even more restricted setting, we can now show the following theorem.

Theorem 6.6.
Let F be a punctured pseudorandom function based on a subversion-resilient PRF *that has its puncture-positions* punc *chosen by a* PRG *that is pseudorandom under subversion.*
Then, F is subversion-resilient, if punc *is fixed throughout the initialization of F or if* punc *gets subversion-resiliently rerandomized right before use.*

Proof. Let F be as described and $\mathcal{A}$ be the adversarial user. We handle both possible cases of punc separately.

1. Let punc be fixed throughout the initialization of F. This means, that punc gets chosen once by the subversion-resilient PRG and can not be changed until after it is used in the puncturing of key k. As such, an adversary $\mathcal{A}$ can not rely on rejection sampling, as this would change punc. Thus, $\mathcal{A}$ can only rely on luck.

In each initialization of the PPRF one set punc is chosen. This set encodes the adversaries current bit with probability $\frac{2^{m-1}-\frac{\binom{m}{m/2}}{2}}{2^m}$. For a message with length $\mathsf{poly}(\lambda)$, $\mathcal{A}$ would need $\mathsf{poly}(\lambda)$ consecutive sets to be chosen correctly. We can describe this probability with the following term:

$$\begin{aligned}\Pr_{\mathsf{punc}}[\mathcal{A}\text{ wins}] &= \prod_{i=0}^{\mathsf{poly}(\lambda)} \frac{2^{m-1}-\frac{\binom{m}{m/2}}{2}}{2^m} \\ &\leq \prod_{i=0}^{\mathsf{poly}(\lambda)} \frac{1}{2} \\ &= \frac{1}{2}^{\mathsf{poly}(\lambda)} = \frac{1}{2^{\mathsf{poly}(\lambda)}} = \mathsf{negl}(\lambda)\end{aligned}$$

This means, at best, $\mathcal{A}$ has a negligible probability of success. The PPRF must then be subversion-resilient.

2. Let punc get subversion-resiliently rerandomized right before use. This must mean, that any modification to punc via rejection sampling by $\mathcal{A}$ only impacts the rerandomization step and not the puncturing itself.
   The algorithm for rerandomization may be subversion-resilient by assumption, however, the output will still encode the adversaries current bit with probability $\frac{2^{m-1}-\frac{\binom{m}{m/2}}{2}}{2^m}$.
   The analysis of case 2 from this point forward is analogous to the analysis of case 1. Thus, the resulting PPRF must be subversion-resilient as well.

The theorem follows from the results of cases 1 and 2. □

We proved that there exist a PPRF that is subversion-resilient, but only under the very strict assumption that we can either force an adversary to use a specific generated randomness or can successfully rerandomize under subversion. When considering these two assumptions it is not clear if they are realistic or not. Thankfully, we can leverage prior research from the fields of algorithm substitution attacks and steganography.
Rerandomization, as we use it in this thesis, is widely understood as one possible implementation of a cryptographic reverse firewall [BBD+23, MSD15, CMNV22, AMV15]. Reverse firewalls use rerandomization and similar tactics to break subversions in *bad* pseudorandomness. While generally considered as a fix to maliciously generated pseudorandomness, its methods also help in our use-case, as the rerandomization voids the usability of the adversaries rejection sampling.
Similarly, our assumption that we can fix set punc prior to execution, is a usage of self-

guarding schemes [FM18]. Self guarding schemes assume a secure, non-subverted initialization phase, in which we can commit to values that we want to use later. While not as widely used as reverse firewalls, the existence of such schemes are considered a valid assumption. As both cryptographic reverse firewalls and self-guarding schemes provide some sound reasoning for our assumption, we consider our result as usable in more complex constructions.
With this, we tackled all necessary primitives to achieve suvbersion-resilient IND-CCA encryption. While we proved that arbitrary PPRFs can not be subversion-resilient we were able to show that we can construct a subversion-resilient PPRF using a random puncturing under strict assumptions. As mentioned before, the Sahai and Waters construction builds upon random puncturing, meaning this assumption would have been necessary either way. The additional assumptions of an existing cryptographic firewall or self-guarding scheme implementation also do not conflict with any of the other used assumptions. As such, the only thing left to do is to combine all our findings.

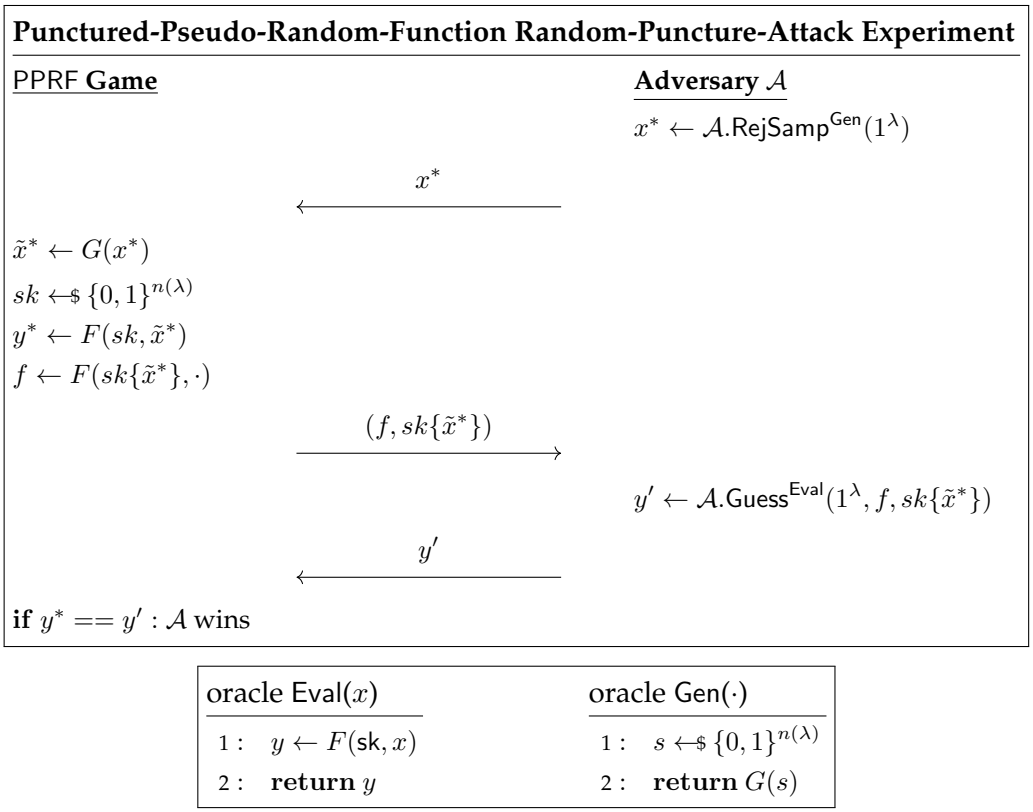
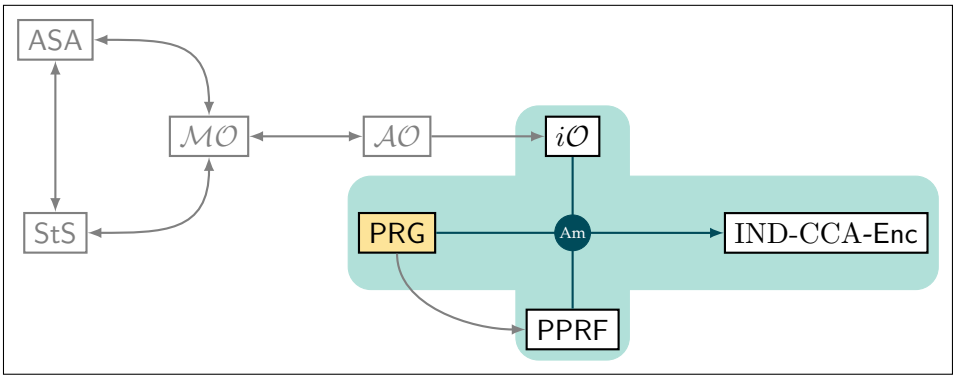


Figure 6.4: A modified version of the PPRF security game from Figure 2.9. Instead of arbitrarily choosing the puncturings punc, $\mathcal{A}$ is only allowed to query a subversion-resilient PRG G to choose punc. The game then, additionally, rerandomizes the adversaries choice before using it to puncture F.

7 Constructing a Subversion-Resilient, CCA-Secure Encryption Scheme



Prior to this chapter, we proved that $\mathcal{AO}$ schemes are subversion-resilient, that subversion-resilient $i\mathcal{O}$ can be build from $\mathcal{AO}$, that cryptographic PRGs are subversion-resilient and that some PPRF schemes are subversion-resilient as well. In this chapter, we will combine all these results to achieve subversion-resilient IND-CCA-secure encryption.
We will use the construction from Sahai and Waters [SW14] to build a IND-CCA-secure encryption scheme $\mathcal{E}$ using only subversion-resilient primitives. By assuming a secure amalgamation function Am in combination with the split programming model, we will then show that $\mathcal{E}$ is subversion-resilient as well. We will do so by following the standard security proof of the construction. However, we will source the security from the subversion-resilience of the constructions subprocedures instead of the standard security assumptions.
Before diving into the subversion-analysis, we will thus provide a short overview of the construction, as well as its security proof.

7.1 Construction Overview

Sahai and Waters provided with [SW14] the first successful attempt at leveraging indistinguishability obfuscation for cryptographic constructions. Prior to their paper, usage of $i\mathcal{O}$ was considered, but not well understood [BGI+12, GR14]. Sahai and Waters realized that the key for using indistinguishability obfuscation was to construct obfuscated programs that behave with overwhelming probability identically. This introduces a negligible error

factor where *everything* can break, but as a trade off provides an additional way to disable adversary oracles. Using game hopping strategies as described in [BR06] it is then possible to iteratively disable *all* adversary oracles until a game is reached where $\mathcal{A}$ can only guess. If all intermediary steps between the original game and this optimal game are negligibly close in their advantage, then the advantage in the original game must be bound in the cumulative advantage gain of the game chain. Note, that for any constant amount of negligibly close hops made, this cumulative advantage is bound by $\mathsf{negl}(\lambda)$.
The Sahai and Waters construction now uses the IND-CCA-security game as a starting point and adds five intermediary hybrid games in the security proof. The challenge element sent to the adversary is step-by-step modified until the challenge consists of pure randomness and provides no information on the actual challenge element $b \in \{0,1\}$. However, before we can discuss this game hopping proof further, we first need to introduce how exactly the construction from Sahai and Waters works.
The construction consists of three algorithms, an initialization algorithm **Setup**, an encryption algorithm **Encrypt** and a decryption algorithm **Decrypt**. The general idea behind the construction is to initialize an obfuscation of a symmetric encryption scheme with hard-coded keys which anyone can use as the public encryption. The symmetric encryption thereby relies on the use of two punctured pseudorandom functions, which get run on the secret keys, a provided message and some newly generated randomness. The underlying symmetric encryption is described with the procedure CCA-PKE-Encrypt.

func **Setup**(1^λ)

1 : $\mathsf{k}_1 \leftarrow F_1.\mathsf{Gen}(1^\lambda)$
2 : $\mathsf{k}_2 \leftarrow F_2.\mathsf{Gen}(1^\lambda)$
3 : $\mathsf{sk} = (\mathsf{k}_1, \mathsf{k}_2)$
4 : $\mathsf{pk} \leftarrow i\mathcal{O}.\mathsf{Obf}$ (CCA-PKE-Encrypt(msg, r))
5 : **return** $(\mathsf{pk}, \mathsf{sk})$

CCA-PKE-Encrypt(msg, r)

1 : $t \leftarrow G(r)$
2 : $c_1 = t$
3 : $c_2 \leftarrow F_1(\mathsf{k}_1, t) \oplus \mathsf{msg}$
4 : $c_3 \leftarrow F_2(\mathsf{k}_2, c_1 \| c_2)$
5 : **return** $c = (c_1, c_2, c_3)$

func **Encrypt**(pk, msg)

1 : $r \leftarrow\!\!\$\ \{0,1\}^\lambda$
2 : $c \leftarrow \mathsf{pk}(\mathsf{msg}, r)$
3 : **return** c

func **Decrypt**(sk, c)

1 : **if** $c_3 == F_2(\mathsf{k}_2, c_1 \| c_2)$:
2 : $\quad \mathsf{msg}' \leftarrow F_1(\mathsf{k}_1, c_1) \oplus c_2$
3 : $\quad$ **return** msg′
4 : **else return** reject

Figure 7.1: The three functions of the Sahai and Waters construction for IND-CCA-secure encryption from $i\mathcal{O}$. **Setup** initializes the encryption scheme $\mathcal{E}$, **Encrypt** is used to encrypt messages and **Decrypt** is used to decrypt received ciphers.

Setup then works by generating the secret key sk as k_1 and k_2 for the PPRFs and constructing the obfuscation of CCA-PKE-Encrypt, with k_1 and k_2 set as constants, as the public

key pk. **Encrypt** works by generating some randomness r and then executing pk on it as well as a 1-bit message msg. Finally, **Decrypt** works by first testing if the received cipher c is sound and, depending on the result, either restores msg or rejects the cipher. We provide pseudocode for the three functions in Figure 7.1.
With this, we can now continue with the discussion of the security proof. As stated before, Sahai and Water use 5 hybrid security games to transform the IND-CCA-security game into a game where the adversary is provided no information. For each hybrid game they show that the advantage of an adversary must be negligibly close to the prior step. For any two consecutive hybrid games Hyb_i and Hyb_{i+1}, this means that

$$\left|\mathsf{Adv}^{\mathsf{Hyb}_i}_{\mathcal{A},\mathcal{E}}(\lambda) - \mathsf{Adv}^{\mathsf{Hyb}_{i+1}}_{\mathcal{A},\mathcal{E}}(\lambda)\right| \leq \mathsf{negl}(\lambda) \quad .$$

We will provide a short description of the changes introduced in each hybrid game, as well as an intuition on how Sahai and Waters prove that the hybrids are stepwise negligibly close. A visual representation of the game hops can be found in Figure 7.2.

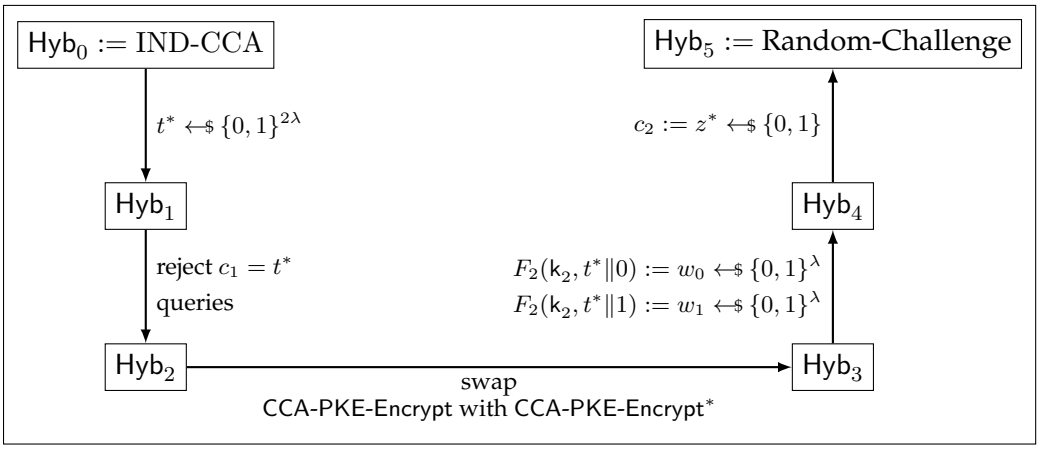


Figure 7.2: Visual representation of the changes between the hybrid games in the security proof of the Sahai and Waters construction. The game hopping starts at Hyb_0, the standard IND-CCA-security game, and ends at Hyb_5, where the adversary $\mathcal{A}$ is provided a challenge that consists only of random elements and no information about the actual challenge bit b. Each *hop* adds at most negligible advantage, meaning the advantage of $\mathcal{A}$ in the IND-CCA game is negligible.

The hybrid games start with Hyb_0, which is simply the standard IND-CCA-security game, as depicted in Figure 2.10. Note that the choice of msg_0 and msg_1 must always be 0 and 1, as the construction only allows for 1-bit messages to be encrypted.
In hybrid game Hyb_1 we change the choice of t^*, the output of the PRG G in CCA-PKE-Encrypt when generating the challenge cipher c^*, to be real randomness instead of a pseudorandomly generated value. In the standard IND-CCA game, the adversary $\mathcal{A}$ is gener-

ally allowed to provide the seed r^*, which is used to generate t^*. With this hybrid game we thus weaken the adversaries impact on the scheme. From the security of the pseudo-random generator G we then get, that Hyb_1 is negligibly close in advantage to the original game.
For hybrid game Hyb_2, we restrict the decryption oracle of $\mathcal{A}$. Any cipher generated by the Sahai and Waters construction consists of three parts c_1, c_2 and c_3. The value c_1 is, thereby, set to the used output of G. In the challenge c^* the first value c_1^* is thus set to t^*. We do not want the adversary to query ciphers that use the value of our challenge. For this, we simply modify the decryption oracle to output $\perp$ when it is queried with $c_1 = t^*$ during phase 1, meaning before $\mathcal{A}$ is given the challenge c^*. As the adversary is only allowed to query their decryption oracle bound in some polynomial $q(\lambda)$, they only notice this modification with a negligible probability. Thus the game is negligibly close to Hyb_1.
The next hybrid game is Hyb_3. In this hybrid, we use the properties of the used indistinguishability obfuscator to swap CCA-PKE-Encrypt with CCA-PKE-Encrypt*. The pseudocode of the modified procedure can be found in Figure 7.3. The modified procedure only differs in the use of punctured keys for the used PPRF functions F_1 and F_2. However, as the punctured positions are based on t^*, which by modification in Hyb_1 is a random value, this means that the two functions only differ in their behavior if t^* is in the domain of G. This only happens with a negligible probability, meaning the input-output behavior does not change with overwhelming probability. From the security of the $i\mathcal{O}$ scheme it then follows that Hyb_3 is negligibly close to Hyb_2.

CCA-PKE-Encrypt(msg, r)		CCA-PKE-Encrypt*(msg, r)
1 : $t \leftarrow G(r)$		1 : $t \leftarrow G(r)$
2 : $c_1 = t$	changes to	2 : $c_1 = t$
3 : $c_2 \leftarrow F_1(\mathsf{k}_1, t) \oplus \mathsf{msg}$	→	3 : $c_2 \leftarrow F_1(\mathsf{k}_1\{t^*\}, t) \oplus \mathsf{msg}$
4 : $c_3 \leftarrow F_2(\mathsf{k}_2, c_1 \| c_2)$		4 : $c_3 \leftarrow F_2(\mathsf{k}_2\{t^* \| 0, t^* \| 1\}, c_1 \| c_2)$
5 : **return** $c = (c_1, c_2, c_3)$		5 : **return** $c = (c_1, c_2, c_3)$

Figure 7.3: Highlighting of the differences between the procedures CCA-PKE-Encrypt and CCA-PKE-Encrypt*. The procedures only differ in the puncturings of keys k_1 and k_2.

In hybrid game Hyb_4 we continue changing parts of the challenge c^* with randomness. The second part of the challenge c_3^* is replaced with a randomly chosen value, in place of the original value $F_2(\mathsf{k}_2, c_1^* \| c_2^*) = F_2(\mathsf{k}_2, t^* \| c_2^*)$. This modification is also made for the decryption oracle of $\mathcal{A}$. From the security of the PPRF F_2 we then get that Hyb_4 is negligibly close to Hyb_3.
Lastly, in hybrid game Hyb_5, we further replace the value of c_2^* with a randomly chosen

$z^* \in \{0,1\}$. The security of the PPRF F_1 guarantees us that Hyb_5 is negligibly close in advantage to Hyb_4. Further, any challenge c^* in Hyb_5 only contains random values, meaning no information on b is disclosed to $\mathcal{A}$. It then follows that the advantage in hybrid game Hyb_5 must be 0, as $\mathcal{A}$ can do nothing but guess at random. When we now assume the worst case advantage gain in each hybrid step, then each step adds $\mathsf{negl}(\lambda)$ to the cumulative advantage. However, for the original IND-CCA game, we can then describe the adversaries advantage with

$$\begin{aligned}
\mathsf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\mathcal{E}}(\lambda) &= \mathsf{Adv}^{\mathsf{Hyb}_0}_{\mathcal{A},\mathcal{E}}(\lambda) \\
&\leq \mathsf{Adv}^{\mathsf{Hyb}_5}_{\mathcal{A},\mathcal{E}}(\lambda) + \sum_{i=0}^{4} \left| \mathsf{Adv}^{\mathsf{Hyb}_\mathrm{i}}_{\mathcal{A},\mathcal{E}}(\lambda) - \mathsf{Adv}^{\mathsf{Hyb}_{\mathrm{i}+1}}_{\mathcal{A},\mathcal{E}}(\lambda) \right| \\
&\leq 0 + \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda) \\
&= 5 \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda) \quad .
\end{aligned}$$

As the adversary was chosen arbitrary it then follows that the insecurity of the encryption scheme in the IND-CCA game is also negligible, finishing the proof from Sahai and Waters. For the subversion-analysis, we will need to reconstruct this proof from different security guarantees. Sahai and Waters use the standard cryptographic security guarantees, however, as we will work on possibly subverted primitives, we can not be sure that standard security holds. Instead, we will need to source the security of the scheme from the subversion-resilience of its parts.
To finalize our proof that the Sahai and Waters construction is an IND-CCA-secure encryption scheme under subversion, we will need to show two things. First, we will need to show that each of the three algorithms composing the scheme is on their own subversion-resilient. Afterwards, we will need to show that we still achieve IND-CCA-security, even when we can only source our security from the subversion-resilience. For this, we will work in the trusted amalgamation with split programming model, as the two models are a tried and proven framework for realistic subversion-analysis.
We will prove both steps in the next section.

7.2 Subversion-Resilient, CCA-Secure Encryption

In this section we will construct the first subversion-resilient IND-CCA-secure encryption scheme. As stated in the last section, there are two steps that we need to prove. First, we need to show that all individual algorithms of the construction are subversion-resilient. Afterwards, we then need to show that we do not break the security proof by sourcing the security from the subversion-resilience of its parts rather than standard security assump-

tions.
As mentioned before, throughout this section we will work exclusively in the trusted amalgamation with split programming model. The trusted amalgamation model provides an amalgamation function Am, which we can leverage to force algorithms to follow their specification. The amalgamation function Am can be understood as *gluing* together blocks of code and forcing them to be executed in fixed order [RTYZ16, RTYZ17]. The split programming model then provides an additional framework in which constructions are allowed to be broken down into a fixed amount of subprocedures, which all can be checked for subversions individually. Further, using trusted amalgamation and split programming we can also guarantee a non-subverted source of *good* randomness, as Russel et al. showed in [RTYZ16].
We can now start with the first of our two steps, showing that there exist subversion-resilient variants for the three algorithms Setup, Encrypt and Decrypt of the Sahai and Waters construction. We start with the Setup construction.

Lemma 7.1.
Let $i\mathcal{O}$ be a subversion-resilient indistinguishability obfuscation scheme and F_1, F_2 be subversion-resilient PPRF. *Then algorithm* Setup *is subversion-resilient in the trusted amalgamation with split programming model.*

Proof. Setup works by first calculating two keys for the PPRFs F_1 and F_2. By assumption, F_1 and F_2 are subversion-resilient, thus each generation of a punctured key must also be subversion-resilient. Otherwise, you could break the subversion-resilience of F_1 and F_2 by only using subverted keys. Combining the keys to sk is assumed a secure operation and thus subversion-resilient as well.
Next, the public key is built by obfuscating the function CCA-PKE-Encrypt. As CCA-PKE-Encrypt is structurally fixed before execution of Setup, and is only modified by adding the two generated punctured keys, any subversion can only be introduced by the obfuscation itself. However, as $i\mathcal{O}$ is assumed subversion-resilient, the resulting obfuscation pk must be subversion-resilient as well.
Combining sk and pk is, again, done using an assumed secure operation. Any individual operation is thus subversion-resilient. Further, using an amalgamation function Am in the split programming model then guarantees us that these operations are executed in the correct order.
It follows, that Setup is subversion-resilient. □

We have shown that there exists a subversion-resilient construction of Setup, assum-

ing subversion-resilient $i\mathcal{O}$ and PPRF exist. In Chapter 5 we showed how to construct subversion-resilient $i\mathcal{O}$ and in Chapter 6 how to construct subversion-resilient PPRF, meaning this construction is sound. We continue with the proof for Encrypt.

Lemma 7.2.
Let G be a subversion-resilient PRG*, $i\mathcal{O}$ be a subversion-resilient indistinguishability obfuscation scheme and F_1, F_2 be subversion-resilient* PPRF*. Assuming there exist secure concatenation and xor operations, then algorithm* Encrypt *is subversion-resilient in the trusted amalgamation with split programming model.*

Proof. The algorithm Encrypt works by first sampling a random value r and then executing pk on said randomness and a message msg. From the split programming model we know that we can securely sample *good* randomness, so generating r is subversion-resilient. The public key pk is a subversion-resilient obfuscation, meaning on execution its subversion-resilience reduces to the subversion-resilience of the obfuscated program. In this case, this is CCA-PKE-Encrypt.
CCA-PKE-Encrypt works by calling a pseudorandom generator G once on the provided randomness r. Afterwards a ciphertext is constructed by using secure operations and calling the two PPRFs F_1 and F_2. We know G, F_1 and F_2 to be subversion-resilient by assumption, meaning the only attack vector against the result would be to maliciously modify the control flow of the program. However, as we work in the trusted amalgamation model, we assume that the program is *glued* together, meaning when CCA-PKE-Encrypt is called its control flow can not be changed.
The execution of pk to generate c is thus subversion-resilient. Using the trusted amalgamation function Am to fix the control flow of Encrypt itself then yields that Encrypt itself is subversion-resilient as well. □

We have shown that there exists a subversion-resilient construction of Encrypt, assuming subversion-resilient PRG, $i\mathcal{O}$ and PPRF. Note that subversion-resilient variants of these three cryptographic primitives have been shown by us to exist in Chapters 5 and 6. Further note that our proof heavily relies on us working in the amalgamation model. If we do not guarantee a fixed control flow, then attacks like rejection sampling could easily break any subversion-resilience guarantees.
As an example, the output c_1 is directly sourced from a subversion-resilient PRG. An attacker that is allowed to rejection sample on pk would then be able to construct a steganographic channel similar to the construction we showed in the proof of Lemma 6.5. Prior,

we fixed the existence of this steganographic channel by rerandomizing the PRG output before usage. Note that we are unable to do so in Encrypt without breaking the soundness of the resulting ciphertext.
We end the first step by showing that the construction of Decrypt is subversion-resilient under sound assumptions.

Lemma 7.3.
Let F_1, F_2 be subversion-resilient PPRF. *Assuming there exist secure compare, concatenation and xor operations, then algorithm* Decrypt *is subversion-resilient in the trusted amalgamation with split programming model.*

Proof. Decrypt works internally by only evaluating subversion-resilient PPRFs and using assumed secure operations on their output. Each individual step is thus subversion-resilient under our assumptions. Using an amalgamation function Am in the split programming model then further guarantees us that these steps are executed once in the correct order.
It follows then that Decrypt is subversion-resilient. □

We have shown that there exists a subversion-resilient construction of Decrypt, assuming subversion-resilient PPRF. With this, we have shown that not only are there subversion-resilient variants of all three algorithms used in the Sahai and Waters construction, but they only rely on cryptographic primitives for which we have shown that subversion-resilient constructions exist. With this, we finished the first of the two steps. It remains to show that we can still leverage the subversion-resilience to achieve a negligible advantage in the IND-CCA-security game. We will show just that in the following Theorem 7.4.

Theorem 7.4.
Let G be a subversion-resilient PRG, *$i\mathcal{O}$ be a subversion-resilient indistinguishability obfuscation scheme and F_1, F_2 be subversion-resilient* PPRF. *Assuming there exist secure compare, concatenation and xor operations, then the Sahai and Waters construction for* IND-CCA*-secure encryption is subversion-resilient in the trusted amalgamation with split programming model.*

Proof. Let $\mathcal{E} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ describe the encryption scheme from the Sahai and Waters construction. We want to show that $\mathcal{E}$ is IND-CCA-secure under subversion. As we work in the subversion setting we assume *all* used algorithms and cryptographic

primitives to be subverted. As such, we can not source security guarantees directly from standard assumptions, but rather need to source them from subversion-resilience instead. Our goal is then to reconstruct the logic chain of the original IND-CCA-security proof with our changed source of security.
When building a security proof from subversion-resilience we need to be sure that we only use subversion-resilient building blocks. As we work in the trusted amalgamation with split programming model, we are provided with a set of subroutines which were used when building $\mathcal{E}$. This set consists of the three algorithms $\mathsf{Setup}, \mathsf{Encrypt}$ and $\mathsf{Decrypt}$, as well as the implementations of the cryptographic primitives used in the algorithms, namely $G, i\mathcal{O}, F_1$ and F_2. By assumptions, we know $G, i\mathcal{O}, F_1$ and F_2 to be subversion-resilient and from Lemma 7.1, 7.2 and 7.3 we know that the three algorithms building $\mathcal{E}$ are under our assumptions subversion-resilient as well.
This means there exist watchdogs Wa for all subprocedures of $\mathcal{E}$ that either detect subversions or guarantee that any subversion has only a negligible probability of being successfully exploited. To guarantee that we only use secure building blocks we can then build a watchdog $\mathsf{Wa}_{\mathcal{E}}$ for the encryption scheme $\mathcal{E}$ that executes the watchdogs for all of the subprocedures. If any watchdog triggers, meaning they can not guarantee subversion-resilience, $\mathsf{Wa}_{\mathcal{E}}$ triggers as well and stops the execution of $\mathcal{E}$ immediately. Conversely, if no watchdog detects a subversion, then $\mathsf{Wa}_{\mathcal{E}}$ accepts the implementation and allows $\mathcal{E}$ to continue. As we only execute $\mathcal{E}$ when none of the watchdogs fails, we then know that we only work with subversion-resilient building blocks.
We can now reconstruct the original IND-CCA-security proof from Sahai and Waters in the subversion setting. For this, we will show that the hybrid games Hyb_i and Hyb_{i+1} are still pairwise negligibly close for $i \in \{0, 1, 2, 3, 4\}$ under subversion. Afterwards, we show that the advantage of Hyb_5, $\mathsf{Adv}^{\mathsf{Hyb}_5}_{\mathcal{A},\mathcal{E}}(\lambda)$, must still be 0 in the subversion setting. This then proves IND-CCA-security under subversion.

Game Hop 1: Hyb_0 to Hyb_1
In the first game hop from Hybrid Hyb_0 to Hyb_1, the generation of t^* is switched from a subversion-resilient PRG G to a uniformly random sample. The value of t^* is returned as part of the challenge cipher, namely c^*_1. Let us now assume that the two games are not negligibly close, meaning there exists a distinguisher D with non-negligible advantage

$$\mathsf{Adv}^{\mathrm{dist}}_{\mathsf{D},\mathsf{Hyb}_0,\mathsf{Hyb}_1}(\lambda) = \left|\mathsf{Adv}^{\mathsf{Hyb}_0}_{\mathcal{A},\mathcal{E}}(\lambda) - \mathsf{Adv}^{\mathsf{Hyb}_1}_{\mathcal{A},\mathcal{E}}(\lambda)\right| \quad .$$

The existence of D now directly imply a steganographic channel using G. An adversary $\mathcal{A}$ sends either outputs of G for a 0 or randomness for a 1. The receiver is now able to reconstruct the message using distinguisher D by simulating $\mathcal{E}$. For a given challenge r,

the receiver constructs c^* with $c_1^* = r$ and provides this as input to D. If r is the output of G, the distinguisher outputs Hyb_0, otherwise Hyb_1, thereby breaking the subversion-resilience of G. As we assume G to be subversion-resilient, our assumption must have been wrong.
Thus, the hybrid games Hyb_0 and Hyb_1 must be negligibly close in advantage.

Game Hop 2: Hyb_1 to Hyb_2
In the second game hop from Hybrid Hyb_1 to Hyb_2 we only modify the decryption oracle of $\mathcal{A}$ in phase 1, meaning before $\mathcal{A}$ receives c^*. We do not answer decryption queries where $c_1 = c_1^*$. Note, that $\mathcal{A}$ trivially detects such a modification in phase 2 after receiving c^*. However, in phase 1 the adversary only detects this change, if and only if they query the oracle exactly with $c_1 = c_1^*$. We allow any given adversary $\mathcal{A}$ to query the decryption oracle $q(\lambda)$ times, where $q(\lambda)$ is some polynomial. Thus, $\mathcal{A}$ detects the modification with probability $\frac{q(\lambda)}{2^\lambda} = \mathsf{negl}(\lambda)$.
The games must then be negligibly close in advantage.

Game Hop 3: Hyb_2 to Hyb_3
In the third game hop from Hybrid Hyb_2 to Hyb_3 we switch out the program obfuscation of $\mathsf{CCA\text{-}PKE\text{-}Encrypt}$ against the obfuscation of $\mathsf{CCA\text{-}PKE\text{-}Encrypt}^*$. The only difference is that the new program has punctured out values in the used keys, namely t^* for k_1 and $t^*\|0$ and $t^*\|1$ for k_2. Note that the two programs thus only differ in execution if the generated value t is equal to t^*.
However, with overwhelming probability, the value t^* is not part of the image of G, as G only maps 2^λ of the $2^{2\lambda}$ values t^* could have been chosen from. This means that, with almost certainty, the punctured values can never naturally occur in the obfuscation. It follows then that, even though F_1 and F_2 are defined differently, with overwhelming probability, the two programs behave identically. We are thus allowed to swap out the two programs without violating the conditions for $i\mathcal{O}$.
It remains to show that the two games are negligibly close in advantage. For that, let us now assume the inverse, namely that there exists a distinguisher D with non-negligible advantage

$$\mathsf{Adv}^{\mathrm{dist}}_{\mathsf{D},\mathsf{Hyb}_2,\mathsf{Hyb}_3}(\lambda) = \left|\mathsf{Adv}^{\mathsf{Hyb}_2}_{\mathcal{A},\mathcal{E}}(\lambda) - \mathsf{Adv}^{\mathsf{Hyb}_3}_{\mathcal{A},\mathcal{E}}(\lambda)\right| \quad .$$

The existence of D now directly implies a steganographic channel using $i\mathcal{O}$. An adversary $\mathcal{A}$ uses either the obfuscation of $\mathsf{CCA\text{-}PKE\text{-}Encrypt}$ to send a 0 or $\mathsf{CCA\text{-}PKE\text{-}Encrypt}^*$ to send a 1. The receiver is now able to reconstruct the message using distinguisher D by simulating $\mathcal{E}$. For a given challenge ch, the receiver simulates $\mathcal{E}$ and provides ch as the challenge cipher to D. If ch is the output of $\mathsf{CCA\text{-}PKE\text{-}Encrypt}$, the distinguisher outputs Hyb_2, otherwise Hyb_3, thereby breaking the subversion-resilience of $i\mathcal{O}$. As we

assume $i\mathcal{O}$ to be subversion-resilient, our assumption must have been wrong.
The two hybrid games must thus be negligibly close in advantage.

Game Hop 4: Hyb_3 to Hyb_4
In the fourth game hop from Hyb_3 to Hyb_4 we replace the evaluation of $F_2(\mathsf{k}_2, t^* \| 0)$ and $F_2(\mathsf{k}_2, t^* \| 1)$ with randomly chosen values. Note that at this point t^* is, following our initial game hop from Hyb_0 to Hyb_1, chosen at random and thus outside of the attackers control. Further note that one of these two values will be part of the challenge c^*, while the other can only be part of a phase 2 oracle query.
Assume now that there exists a distinguisher with non-negligible advantage

$$\mathsf{Adv}^{\mathrm{dist}}_{\mathsf{D},\mathsf{Hyb}_3,\mathsf{Hyb}_4}(\lambda) = \left|\mathsf{Adv}^{\mathsf{Hyb}_3}_{\mathcal{A},\mathcal{E}}(\lambda) - \mathsf{Adv}^{\mathsf{Hyb}_4}_{\mathcal{A},\mathcal{E}}(\lambda)\right| \quad .$$

The existence of D then directly implies a steganographic channel using F_2. An adversary $\mathcal{A}$ samples t^* and sends it either in combination with $r = F_2(k_2, t^* \| b)$ to send a 0 or $r^* \leftarrow\!\!\$\ \{0,1\}^\lambda$ to send a 1. The receiver is now able to reconstruct the message using distinguisher D by simulating $\mathcal{E}$. When the receiver gets the two values they construct c^* from it with $c_1 = t^*$ and $c_3 = r$ and provide it as input to D. Depending on the input, D either outputs Hyb_3 or Hyb_4, breaking the subversion-resilience of F_2. As we assume F_2 to be subversion-resilient, our assumption must have been wrong.
The games must thus be negligibly close in advantage.

Game Hop 5: Hyb_4 to Hyb_5
In the last game hop from Hyb_4 to Hyb_5 we swap the output of $F_1(\mathsf{k}_1, t^*) \oplus \mathsf{msg}$ with a random bit. With functionally the same argument as in the last game hop there can not be a distinguisher D between Hyb_4 and Hyb_5, as this D would allow for a steganographic channel using F_1, breaking its subversion-resilience.
The games Hyb_4 and Hyb_5 must then, again, be negligibly close in advantage.

With this we showed that each two consequitive hybrid games are negligibly close to each other. From this we get that we can bound the advantage of the IND-CCA game under subversion with

$$\begin{aligned}
\mathsf{Adv}^{\text{sub-res IND-CCA}}_{\mathcal{A},\mathcal{E}}(\lambda) &= \mathsf{Adv}^{\mathsf{Hyb}_0}_{\mathcal{A},\mathcal{E}}(\lambda) \\
&\leq \mathsf{Adv}^{\mathsf{Hyb}_5}_{\mathcal{A},\mathcal{E}}(\lambda) + \sum_{i=0}^{4}\left|\mathsf{Adv}^{\mathsf{Hyb}_i}_{\mathcal{A},\mathcal{E}}(\lambda) - \mathsf{Adv}^{\mathsf{Hyb}_{i+1}}_{\mathcal{A},\mathcal{E}}(\lambda)\right| \\
&\leq \mathsf{Adv}^{\mathsf{Hyb}_5}_{\mathcal{A},\mathcal{E}}(\lambda) + 5 \cdot \mathsf{negl}(\lambda) \quad .
\end{aligned}$$

Note now that Hyb_5 provides a challenge element $c^* = (c_1^*, c_2^*, c_3^*) = (t^*, z^*, w^*)$ with

$t^* \leftarrow\$ \{0,1\}^{2\lambda}$, $z^* \leftarrow\$ \{0,1\}$ and $w^* \leftarrow\$ \{0,1\}^{\lambda}$. As each element in c^* is sampled uniformly random, this means that the challenge cipher conveys no information about msg. Further, as the split-programming model guarantees us a secure source of randomness, we additionally know that none of the values contain a subversion. This means there can be no adversary with advantage, meaning $\mathsf{Adv}^{\mathsf{Hyb}_5}_{\mathcal{A},\mathcal{E}}(\lambda) = 0$. It then follows that

$$\begin{aligned}\mathsf{Adv}^{\text{sub-res IND-CCA}}_{\mathcal{A},\mathcal{E}}(\lambda) &\leq \mathsf{Adv}^{\mathsf{Hyb}_5}_{\mathcal{A},\mathcal{E}}(\lambda) + 5 \cdot \mathsf{negl}(\lambda) \\ &= 0 + 5 \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda) \quad .\end{aligned}$$

With this we have shown that the Sahai and Waters construction for IND-CCA-secure encryption from $i\mathcal{O}$ is subversion-resilient under our assumptions. □

With this we finished the second step of our proof as well. With this we have shown that the Sahai and Waters construction is not only constructable under subversion, but it keeps its IND-CCA-security guarantees. With this, we have not only shown that IND-CCA-security is achievable under subversion, but also achieved our goal of providing a first construction for it. In the next chapter we conclude our findings and discuss some possible future works.

8 Conclusions

Throughout this thesis, we have shown the first construction of subversion-resilient IND-CCA-secure encryption as well as subversion-resilient constructions for the necessitated cryptographic primitives. We use this chapter to summarize our results along the research questions defined in Chapter 3. For each question, we state a fitting research answer shortly describing the results we achieved. Afterwards, we discuss possible interests of future research in this field.

8.1 Summary

In Chapter 3, we laid out the five research questions of interest to this thesis. For each of them, we defined a goal to reach and provided a short paragraph describing our methods to achieve said goal. In Research Question 1, we asked whether we can show a deeper relation between auditable obfuscation and subversion-resilience. As stated beforehand, we can now positively answer this question with our results from Chapter 4:

Research Answer 1 (On Auditable Obfuscation and Subversion-Resilience).
We have shown a deeper relation between auditable obfuscation and subversion-resilience as auditable obfuscation is an edge-case of subversion-resilience, wherein malicious obfuscation models algorithm substitution attacks against the program class of obfuscation schemes.

Our proof heavily relies on the results from Berndt and Liśkiewicz in [BL17], adapting their results of algorithm substitution attacks being equivalent to the steganographic setting on a certain type of channel into the obfuscation setting.
Research Question 2 asked whether a relation between indistinguishability obfuscation and $\mathcal{AO}$ and, if so, could be leveraged to construct subversion-resilient $i\mathcal{O}$. We were able to sufficiently analyze the relation between the two primitives and can thus answer the question using our results from Chapter 5.

Research Answer 2 (On Subversion-Resilient Indistinguishability Obfuscation).
We have shown that indistinguishability obfuscation can be constructed from auditable obfuscation while maintaining its auditability. By leveraging that the constructed $i\mathcal{O}$ scheme is just auditable obfuscation acting on a subset of its domain, we were able to prove that subversion-resilient indistinguishability obfuscation is achievable.

This result was mostly achieved through proof by construction, as we were able to leverage the inherent similarity between $\mathcal{AO}$ and $i\mathcal{O}$ definitions.
In Research Question 3, we asked if constructing subversion-resilient PRGs is possible. We investigated this in Section 6.1 of Chapter 6 and can now answer as follows:

Research Answer 3 (Subversion-Resilient Pseudorandom Generators).
We demonstrated that subversion-resilient PRGs can be constructed. Furthermore, we have shown that any PRG sufficing the standard cryptographic indistinguishability game for PRGs inherently is subversion-resilient.

Our proof followed a similar idea as the works of Bemmann et al. in [BBD+23], realizing that the security game for PRGs is similar in construction to that of weak-PRFs. This allowed us to adapt Bemmann et al.'s proof of weak-PRFs inherently being subversion-resilient to PRGs. We further realized that their subversion-resilience proof of the Naor-Reingold PRF construction from weak-PRFs [NR95] can similarly be adapted to show that the GGM construction of PRFs from PRGs [GGM86] is subversion-resilient as well. We proved this in Section 6.2.
As our penultimate goal, we asked about the possibility of constructing subversion-resilient PPRFs in Research Question 4. We reply to this question with our results from Section 6.3:

Research Answer 4 (Subversion-Resilient Punctured Pseudorandom Functions).
We have shown that subversion-resilient punctured PRFs are achievable, but only under a strict set of assumptions. We have further proven that more relaxed assumptions *always* allow for an adversary to leverage the puncturing of the PPRF as a steganographic channel.

We started the section on PPRFs using minimal limitations on the adversary and iteratively restricted them as long as we were still able to construct a steganographic channel. While we were able to achieve subversion-resilient PPRFs in the end, we were only able to do so under very strict assumptions.
Lastly, Research Question 5 asked whether we could use the Sahai and Waters construction to achieve a subversion-resilient IND-CCA-secure encryption scheme. We can positively answer this question using our results from Chapter 7.

Research Answer 5 (On Subversion-Resilient IND-CCA-Secure Encryption).
We have shown that the Sahai and Waters construction can be used to construct a subversion-resilient IND-CCA-secure encryption scheme in the trusted amalgamation with split programming model.

This was achieved leveraging the results of the prior chapters in combination with the trusted amalgamation and split-program model. Our proof follows the original security proof in [SW14], but uses subversion-resilience as its source for security.
We were thus able to positively answer all of our research questions, most of them exhaustively, even. In the next section, we discuss where future works can build upon our results, as well as the few statements we left as open remarks.

8.2 Discussion and Future Works

Our results provide the groundwork for future research to build upon. We showed that auditable obfuscation is subversion-resilient, $\mathcal{AO}$ implies subversion-resilient $i\mathcal{O}$, subversion-resilient PRG and subversion-resilient PPRF being achievable, and provided the first subversion-resilient construction of IND-CCA-secure encryption. Each of these is a firm point for future work to continue from.
While we did show that auditable obfuscation describes subversion-resilience over obfuscation schemes, we did not further analyze the primitive. As stated multiple times throughout this thesis, auditable obfuscation still is a rather new concept and should be held up to scrutiny. As such, a deeper dive into auditable obfuscation and its soundness as a cryptographic primitive could be of interest for future works.
For indistinguishability obfuscation, we see three main points of interest. The first, as mentioned in Chapter 5, we can not completely dismiss that some special cases of $i\mathcal{O}$ imply $\mathcal{AO}$. We shortly discuss this in Appendix A, but see potential in further research. Secondly, we only showed that subversion-resilient $i\mathcal{O}$ exists under assumption of $\mathcal{AO}$.

There might be ways of constructing subversion-resilient $i\mathcal{O}$ from weaker assumptions. Trying to find further constructions seems to be a promising avenue of future research to us. Lastly, as $i\mathcal{O}$ is quite versatile in usage, the proof of subversion-resilient $i\mathcal{O}$ existing has created opportunities of proving a multitude of novel subversion-resilient constructions for other cryptographic primitives. Examples include the other constructions from [SW14], such as universal deniable encryption, key encapsulation mechanisms and non-interactive zero-knowledge proofs, or the construction of full-domain hashes from $i\mathcal{O}$ in [HSW14]. More recently Coladangelo and Gunn [CG24] applied $i\mathcal{O}$ to quantum problems; Wang et al. [WWZ25], used $i\mathcal{O}$ to build homomorphic witness encryption; and Zhang et al. [ZHZ+24] employed $i\mathcal{O}$ to achieve self-bilinear maps.
Lastly, while we did show that subversion-resilient IND-CCA-secure encryption is possible, we only achieved such a scheme for 1-bit messages. Further research could be focused on either improving our construction or building a more efficient subversion-resilient IND-CCA-secure encryption scheme. The key encapsulation mechanisms (KEM) of Sahai and Waters seem to be a promising option, as they can be extrapolated to create an IND-CCA-secure encryption scheme for arbitrary long messages.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 364–375, New York, NY, USA, 2015. Association for Computing Machinery.

[AMV20] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signatures: Definitions, constructions and applications. *Theoretical Computer Science*, 820:91–122, 2020.

[And96] Ross J. Anderson. Stretching the limits of steganography. In *Proceedings of the First International Workshop on Information Hiding*, page 39–48, Berlin, Heidelberg, 1996. Springer-Verlag.

[AP22] Marcel Armour and Bertram Poettering. Algorithm substitution attacks against receivers. *International Journal of Information Security*, 21(5):1027–1050, Oct 2022.

[BBC+14] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions. In Yehuda Lindell, editor, *Theory of Cryptography*, pages 26–51, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BBC24] Pascal Bemmann, Sebastian Berndt, and Rongmao Chen. Subversion-resilient signatures without random oracles. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security*, pages 351–375, Cham, 2024. Springer Nature Switzerland.

[BBD+23] Pascal Bemmann, Sebastian Berndt, Denis Diemert, Thomas Eisenbarth, and Tibor Jager. Subversion-resilient authenticated encryption without random oracles. In Mehdi Tibouchi and XiaoFeng Wang, editors, *Applied Cryptography and Network Security*, pages 460–483, Cham, 2023. Springer Nature Switzerland.

[BBG13] James Ball, Julian Borger, and Glenn Greenwald. Revealed: how us and uk spy agencies defeat internet privacy and security. *The Guardian*, Sep. 2013.

[BCJ21] Pascal Bemmann, Rongmao Chen, and Tibor Jager. Subversion-resilient public key encryption with practical watchdogs. In *Public-Key Cryptography – PKC 2021: 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10–13, 2021, Proceedings, Part I*, page 627–658, Berlin, Heidelberg, 2021. Springer-Verlag.

[Ber14] Nick Berry. Impossible escape? `https://datagenetics.com/blog/december12014/index.html`, Dec 2014.

[Ber18] Sebastian Berndt. *New Results on Feasibilities and Limitations of Provable Secure Steganography*. Phd thesis, University of Lübeck, October 2018.

[BG23] Shalini Banerjee and Steven D. Galbraith. Auditable obfuscation. Cryptology ePrint Archive, Paper 2023/1476, 2023.

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 1–18, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2), may 2012.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Proceedings of the 17th International Conference on Public-Key Cryptography — PKC 2014 - Volume 8383*, page 501–519, Berlin, Heidelberg, 2014. Springer-Verlag.

[BGJS16] Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. Verifiable functional encryption. In *Proceedings, Part II, of the 22nd International Conference on Advances in Cryptology — ASIACRYPT 2016 - Volume 10032*, page 557–587, Berlin, Heidelberg, 2016. Springer-Verlag.

[BL17] Sebastian Berndt and Maciej Liśkiewicz. Algorithm substitution attacks from a steganographic perspective. *CoRR*, abs/1708.06199, 2017.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[BPR14] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 1–19, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, page 409–426, Berlin, Heidelberg, 2006. Springer-Verlag.

[BV16] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation: From approximate to exact. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography*, pages 67–95, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, pages 280–300, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Cac04] Christian Cachin. An information-theoretic model for steganography. *Information and Computation*, 192(1):41–56, 2004.

[CCK+22] Ran Canetti, Suvradip Chakraborty, Dakshita Khurana, Nishant Kumar, Oxana Poburinnaya, and Manoj Prabhakaran. Coa-secure obfuscation and applications. In *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part I*, page 731–758, Berlin, Heidelberg, 2022. Springer-Verlag.

[CG24] Andrea Coladangelo and Sam Gunn. How to use quantum indistinguishability obfuscation. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, page 1003–1008, New York, NY, USA, 2024. Association for Computing Machinery.

[CHY20] Rongmao Chen, Xinyi Huang, and Moti Yung. Subvert kem to break dem: Practical algorithm-substitution attacks on public-key encryption. In *Advances in Cryptology – ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Ko-*

rea, December 7–11, 2020, Proceedings, Part II, page 98–128, Berlin, Heidelberg, 2020. Springer-Verlag.

[CMMV25] Suvradip Chakraborty, Lorenzo Magliocco, Bernardo Magri, and Daniele Venturi. Key exchange in the post-snowden era: Universally composable subversion-resilient pake. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 101–133, Singapore, 2025. Springer Nature Singapore.

[CMNV22] Suvradip Chakraborty, Bernardo Magri, Jesper Buus Nielsen, and Daniele Venturi. Universally composable subversion-resilient cryptography. In *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part I*, page 272–302, Berlin, Heidelberg, 2022. Springer-Verlag.

[CV09] Ran Canetti and Mayank Varia. Non-malleable obfuscation. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, page 73–90, Berlin, Heidelberg, 2009. Springer-Verlag.

[DFP15] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In Gregor Leander, editor, *Fast Software Encryption*, pages 579–598, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[FM18] Marc Fischlin and Sogol Mazaheri. Self-guarding cryptographic protocols against algorithm substitution attacks. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 76–90, 2018.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. *How to construct random functions*, volume 33, page 792–807. Association for Computing Machinery, New York, NY, USA, aug 1986.

[GJLS21] Romain Gay, Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from simple-to-state hard problems: New assumptions, new techniques, and simplification. In *Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part III*, page 97–126, Berlin, Heidelberg, 2021. Springer-Verlag.

[GR14] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. *J. Cryptol.*, 27(3):480–505, jul 2014.

[HMLS07] Dennis Hofheinz, John Malone-Lee, and Martijn Stam. Obfuscation for cryptographic purposes. In *Proceedings of the 4th Conference on Theory of Cryptography*, TCC'07, page 214–232, Berlin, Heidelberg, 2007. Springer-Verlag.

[HRSV11] Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. *J. Cryptol.*, 24(4):694–719, oct 2011.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 201–220, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[HvAL09] Nicholas Hopper, Luis von Ahn, and John Langford. Provably secure steganography. *IEEE Transactions on Computers*, 58(5):662–676, 2009.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 60–73, New York, NY, USA, 2021. Association for Computing Machinery.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[KMN+14] Ilan Komargodski, Tal Moran, Moni Naor, Rafael Pass, Alon Rosen, and Eylon Yogev. One-way functions and (im)perfect obfuscation. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 374–383, 2014.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, page 669–684, New York, NY, USA, 2013. Association for Computing Machinery.

[LCW+25] Jiahao Liu, Rongmao Chen, Yi Wang, Xincheng Tang, and Jinshu Su. Subversion-resilient authenticated key exchange with reverse firewalls. In Joseph K. Liu, Liqun Chen, Shi-Feng Sun, and Xiaoning Liu, editors, *Provable and Practical Security*, pages 181–200, Singapore, 2025. Springer Nature Singapore.

[LPS04] Benjamin Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 20–39, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[MSD15] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 657–686, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[NR95] M. Naor and O. Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 170–181, 1995.

[RTYZ16] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 34–64, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[RTYZ17] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 907–922, New York, NY, USA, 2017. Association for Computing Machinery.

[Sha07] Hovav Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In Sabrina De Capitani di Vimercati and Paul Syverson, editors, *Proceedings of CCS 2007*, pages 552–61. ACM Press, oct 2007.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 475–484, New York, NY, USA, 2014. Association for Computing Machinery.

[Wig19] Avi Wigderson. *Mathematics and Computation - A Theory Revolutionizing Technology and Science*. Princeton University Press, Princeton, 2019.

[WWZ25] Yuzhu Wang, Xingbo Wang, and Mingwu Zhang. Homomorphic witness encryption and its applications. *International Journal of Network Management*, 35(1):e2303, 2025. e2303 nem.2303.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227, 1977.

[Yao82] Andrew C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 80–91, 1982.

[YLL+25] Xiaodong Yang, Xilai Luo, Zefan Liao, Wenjia Wang, Xiaoni Du, and Shudong Li. A cp-abe-based access control scheme with cryptographic reverse firewall for iov. *Journal of Systems Architecture*, 160:103331, 2025.

[YY97] Adam Young and Moti Yung. Kleptography: using cryptography against cryptography. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'97, page 62–74, Berlin, Heidelberg, 1997. Springer-Verlag.

[ZGL20] Yuyang Zhou, Jing Guo, and Fagen Li. Certificateless public key encryption with cryptographic reverse firewalls. *Journal of Systems Architecture*, 109:101754, 2020.

[ZHZ+24] Huang Zhang, Ting Huang, Fangguo Zhang, Baodian Wei, and Yusong Du. Self-bilinear map from one way encoding system and io. *Information*, 15(1), 2024.

A From Indistinguishability Obfuscation to Auditable Obfuscation

In Chapter 4.1, we showed that any auditable obfuscation scheme can be instantiated in such a way that it also behaves as an indistinguishability obfuscation scheme for all programs $P \in \mathcal{I}_\lambda$. We did so by proving that simply running $\mathcal{AO}$ on implementations for one singular $P \in \mathcal{I}$ constitutes a subversion-resilient indistinguishability obfuscation scheme.

However, did not answer whether we could construct an $\mathcal{AO}$ scheme from $i\mathcal{O}$. While this does not seem to be the case at first glance, as we do not force the verifiability property in $i\mathcal{O}$, this is not argued as easily as one might think.

Let us provide an example case in which this argumentation seems to reach its limits. Let $i\mathcal{O}$ be a secure indistinguishability obfuscator over a set of implementations $\mathcal{I} = \{P\}$ and $\mathcal{MO}$ a malicious obfuscation scheme on $i\mathcal{O}$. Let us now, without loss of generality, assume that $\mathcal{MO}$ wants to embed two different triggers, depending on whether the underlying implementation fulfills some kind of easily evaluated predicate $\mathbb{P}$ or not.

Next, consider two implementations $I_0, I_1 \in \mathcal{I}$, where $\mathbb{P}[I_0] = 0$ and $\mathbb{P}[I_1] = 1$, in which $\mathcal{MO}$ wants to embed different triggers. However, any embedding by $\mathcal{MO}$ can also be understood as an honest obfuscation of a program that already possessed the introduced trigger. As such, $\mathcal{MO}$ now technically constructs obfuscations of two different programs that should be indistinguishable from the output of $i\mathcal{O}$.

However, $i\mathcal{O}$ does not guarantee any indistinguishability for obfuscations of differing programs. This means that, at most, one of the results $\mathcal{MO}.\mathsf{Emb}(\mathsf{aux}, I_0, \mathsf{ok})$ and $\mathcal{MO}.\mathsf{Emb}(\mathsf{aux}, I_1, \mathsf{ok})$ can generally be indistinguishable from normal outputs of $i\mathcal{O}$. Otherwise, $i\mathcal{O}$ could obfuscate more than one type of program, which we generally assume to be false.

For this rather artificial example, we can now easily create an auditor Au that simply samples two implementations, one that fulfills $\mathbb{P}$ and one that does not. If the auditor can distinguish between the two obfuscation, the underlying scheme is malicious.

While we do not formally prove that this auditor Au has a non-negligible probability of success, we do consider this example evident enough to not give a conclusive judgment on the original statement. As such, we also consider it worthwhile to further investigate whether indistinguishability obfuscation provides some inherent defenses against some classes of malicious obfuscation.

B Efficient Encodings for puncturing-based Stegosystems

In Chapter 6.3 we showed how to use the puncturing of a PPRF to create a steganographic channel. For our proof, we restricted ourselves to a simple 1-bit channel that either sends a 0 or a 1 per each puncturing set. We also restricted ourselves to use only one puncturing position per instantiation. However, in general, the described setting allows for an arbitrary large set punc, meaning we can find exponentially better message encodings. We will discuss four such possibilities in this appendix.

Efficiently Encoding Messages with Arbitrary Puncture-Sets at Fixed Depth

Let us assume that we have a user-defined puncturable PRF F that can be arbitrarily punctured at depth ℓ. Layer ℓ consists of 2^ℓ nodes, which can either be punctured or not. For our original channel construction we only punctured one single node and took their position $i \bmod 2$ as the resulting message. However, generally speaking, there is nothing that stops us from puncturing more positions. We can then leverage this to achieve a trivial encoding, where each position corresponds to a bit in the message, meaning we can send up to 2^{2^ℓ} messages.
For example, let us use $\ell = 3$. This means there are 8 nodes which we can either puncture or not. We can now send an arbitrary message $\mathsf{msg} \in \{0,1\}^8$. The following Figure B.1 shows just such an encoding for $\mathsf{msg} = 01000110$, the ASCII-Encoding of the letter "F".

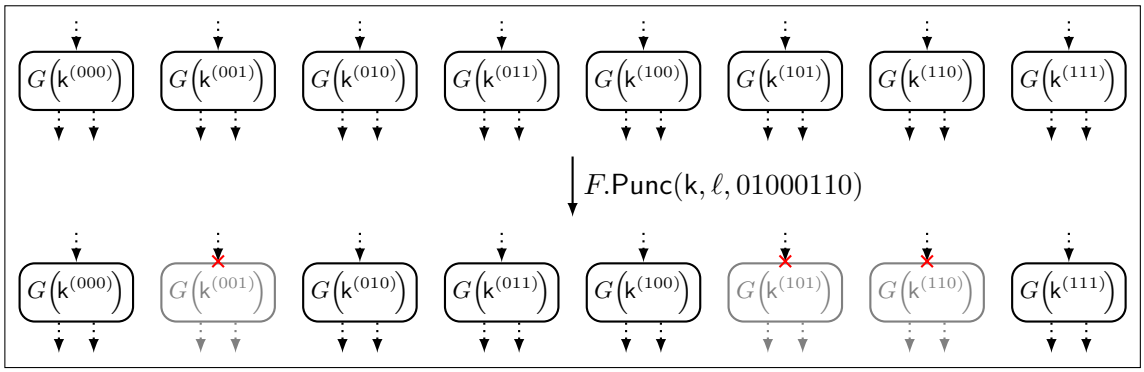


Figure B.1: Picture of layer $\ell = 3$ for the GGM-tree of F before and after puncturing. The version at the bottom of the picture shows an encoding of message 01000110, where a 0 at position i correlates to a non-puncturing at position i and vice versa, a 1 at position i corresponds to a puncturing at position i.

A receiver can then restore the message by extracting the provided set punc from the punctured key $\mathsf{k}\{\mathsf{punc}\}$ and reconstructing the layer. Note that $\ell \in \mathcal{O}(1)$. As such, ℓ will almost certainly describe an inner layer of the construction, as the entire depth of the GGM-tree may be polynomial, meaning the leaf layer is exponential in size. As we only consider a polytime-receiver we thus need to guarantee that we do not reach a layer that would result in exponential reconstruction work.

Efficiently Encoding Messages with 1-Puncturing-Per-Layer Puncture-Sets

Let us again assume that we have a user-defined puncturable PRF F that can be arbitrarily punctured. However, let us now assume that we work in a setting where the above strategy is known to the watchdog, meaning any puncture set with too many puncturings in *one* layer gets flagged as malicious. This would only allow for sparse messages to be sent using the prior encoding. Note, however, that this watchdog would not flag a puncture set as malicious if it only contains at most one puncturing per layer, as such sets are sparse on a per-layer basis.

Let us now assume that F has depth $d \in \mathsf{poly}(\lambda)$. We can now choose a random path through the GGM-tree of F and additionally build the resulting copath for it. The copath of a path W is defined as the set of sibling nodes for all nodes in W. Note, that for each layer, except the root layer, there is exactly one node in the chosen path and exactly one node in the related copath, meaning the size of the copath is d. We can now leverage this copath to send up to 2^d messages.

For example, let us now use $d = 5$. This means there are exactly 5 nodes in the copath set. We can now send an arbitrary messages $\mathsf{msg} \in \{0,1\}^5$. The Figure B.2 shows such an encoding for $\mathsf{msg} = 10110$. Note, that the GGM tree was pruned to simplify the figure.

The sender starts the channel by choosing a path and creating the copath set. Afterwards, for each layer ℓ, the sender looks at bit ℓ of msg. If the relevant bit is a 1, then the node, and all entries in its subtree, are punctured. Otherwise, the sender does nothing and continues with the next layer.

A receiver can now reconstruct the message in the following way. First, the receiver reconstructs set punc from the punctured key $\mathsf{k}\{\mathsf{punc}\}$. Then, the receiver finds the puncturing with maximum depth, for our example in Figure B.2 this would be 1000 in layer 4. As $|\mathsf{punc}| = d$, the receiver is able to do this in polytime. Note, that finding this puncturing fixes the random path up to the layer of the puncturing. Further note, as there is no puncture on a deeper layer, that the end of msg is fixed to a 0-string.

With the path fixed, the receiver must then just check for each node in the path, if the relative node in the copath was punctured or not. For each position, if the node was punctured, the receiver reads a 1, else a 0. At most, this takes $(d-1)^2 = \mathsf{poly}(\lambda)$ work, meaning

the receiver acts in polytime.

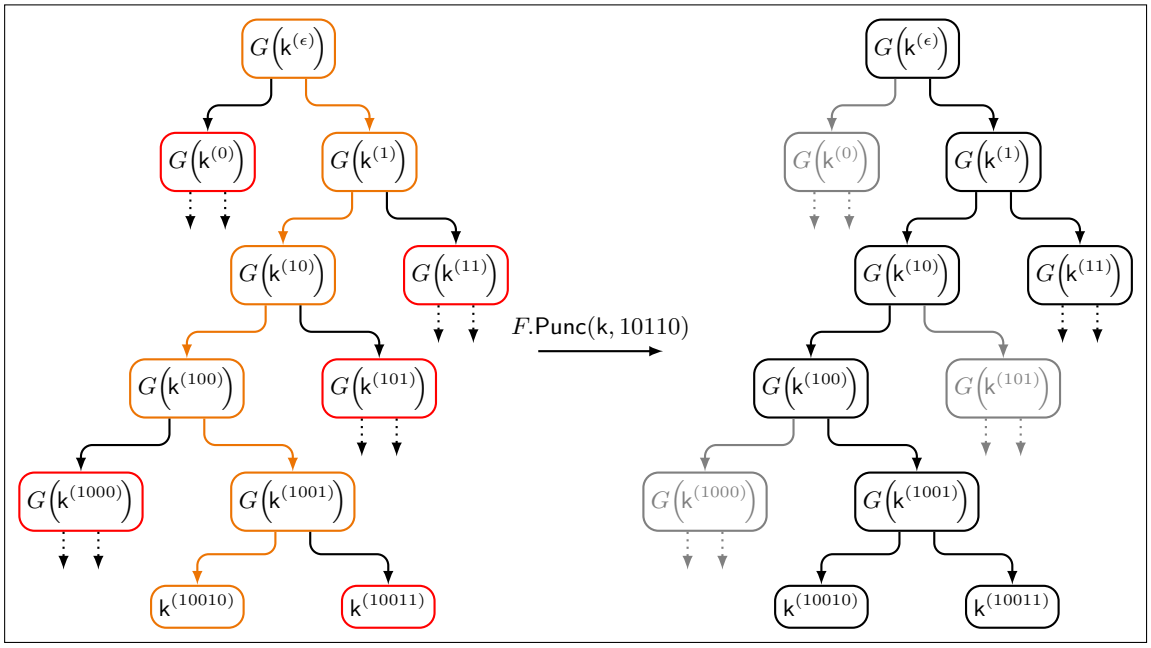


Figure B.2: Picture of a pruned GGM-tree of F before and after puncturing. In the left tree, orange nodes highlight the chosen path through F and red nodes highlight the related copath. In the right tree, we encode message 10110 through the puncturing of the copath.

Efficiently Encoding Messages with 1-Puncturing Puncture-Sets

Let us now assume that we have a user-defined puncturable PRF F that can only be punctured once by the user. Restricting the user in this way makes the prior two encodings impossible, but still keeps the inefficient 1-bit encoding. However, we can improve on this.

Let us assume that we are allowed to puncture F at layer ℓ, with $\ell \in \mathsf{poly}(\lambda)$. We can now send up to 2^ℓ messages with a trivial encoding. Suppose you are allowed to puncture F in layer $\ell = 8$. Layer 8 consists either of $2^8 = 256$ interior nodes, each with their own subtree, or $2^8 = 256$ leaves. Either way, we will work on keys $\mathsf{k}^{(00000000)}$ to $\mathsf{k}^{(11111111)}$, evaluating G on them or returning them as the output of F.

Let us now choose $\mathsf{msg} \in \{0,1\}^8$ arbitrarily. Note that msg has the form $b_1b_2b_3b_4b_5b_6b_7b_8$, meaning we can find the node working on $\mathsf{k}^{(\mathsf{msg})}$ at position msg of layer 8. The sender now punctures this node. The receiver can then reconstruct msg by simply extracting punc from $\mathsf{k}\{\mathsf{punc}\}$, as $\mathsf{punc} = \{\mathsf{msg}\}$. Note that this is polytime, even if the puncturing happens in the leaf layer of a PPRF with depth $\mathsf{poly}(\lambda)$.

Efficiently Encoding Messages with 1-Modification Random Puncture-Sets

In this last section, let us now assume that we have a puncturable PRF F, as well as some existing puncture set punc in fixed layer $\ell \in \mathcal{O}(1)$. Further, lets assume that the user is now only allowed to modify one position in punc, either adding or removing a puncture position. Surprisingly, even this quite restrictive setting allows for an efficient encoding. Precisely, we are able to encode 2^ℓ different messages in layer ℓ. Note, that ℓ may not be polynomial, as we would then have exponentially many nodes in layer ℓ.
We will follow an idea from the blog post [Ber14]. The article describes a prisoners problem, where two prisoners, henceforth named Alice and Bob, need to communicate via a chessboard with a coin on each cell. At the beginning, a jailer randomly places the coins on the board, meaning some amount of coins will show heads while the rest will show tails. Afterwards, the jailer tells Alice one square χ on the board, which Bob needs to guess to win. Alice is then allowed to flip any one coin on the board to tell Bob which square he needs to guess. Only then is Bob allowed to look at the chessboard and choose a square as his answer. If Bob chooses correctly, Alice and Bob are set free.
The winning strategy, as described in [Ber14], is to build a parity code over the coins. A chessboard consists of an 8x8 grid of squares, meaning there are exactly 64, or 2^6, cells. If we now enumerate the cells from 0 to 63, then we can group the cells into 6 sets. We present a visual representation of this grouping in Figure B.3.

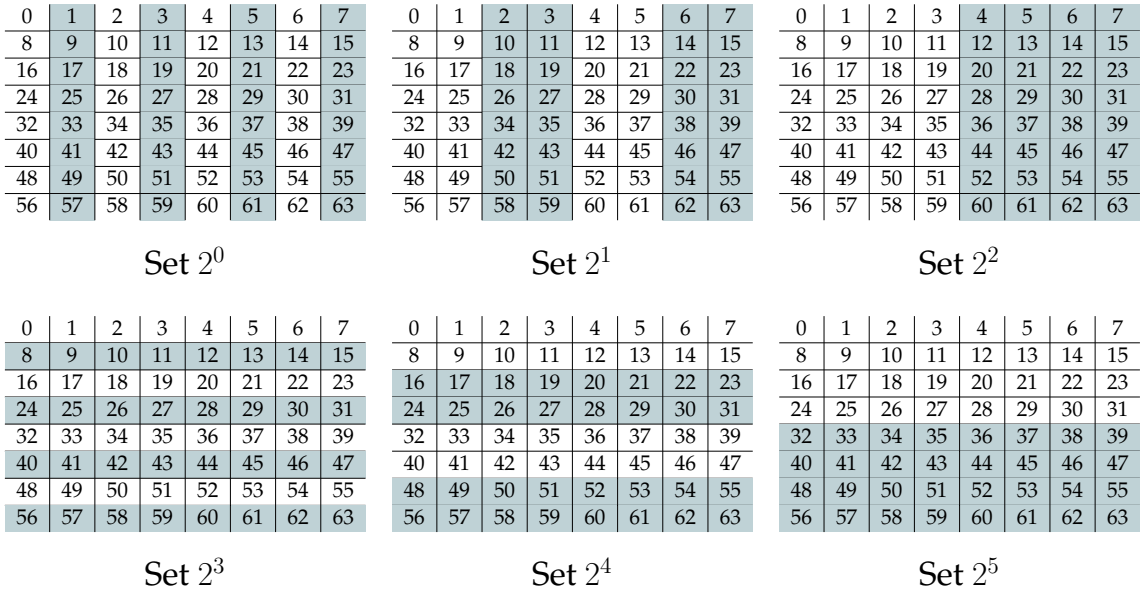


Figure B.3: Parity-sets for the 8x8 grid of a chessboard. For each set the included elements are highlighted.

Each set corresponds to a specific bit-position in the binary representation of a number being set to 1. For example, 42 represented in binary is 101010, meaning it gets added to the sets 2^1, 2^3 and 2^5. Each set then consists of all numbers between 0 and 63 that has the sets respective bit set to 1.
For each set, we can build the xor over all the coins placed on its cells, counting tails as a 0 and heads as a 1. We concatenate the resulting bits of the six sets to a 6-bit number ρ. Notice that this construction means $\rho \in \{0, \dots, 63\}$. This value ρ is also called the natural parity of the board. Note now, that the jailers chosen square χ has a related binary representation in $\{0,1\}^6$. Further, if you take the bitwise xor between ρ and χ, you get another value in $\{0,1\}^6$. The bits of this result are set to 0 wherever the natural parity of the corresponding set is equal to the one of χ and set to 1 where they differ. For example, let $\rho = 111001$ and $\chi = 001011$, then the result would be 110010, meaning the natural parity ρ differs from χ in the sets 2^5, 2^4 and 2^1. By definition of our sets, there is exactly one position on the board that can change this difference, namely the coin at position 110010, or, in decimal, 50. By flipping this one coin, we can set the natural parity ρ equal to χ. Note, that our result is dependent on the fact that the size of the chessboard is an exact power of two and does not work otherwise. Further note, that the size of the set was the *only* dependency we had.
We can now adapt this strategy to our puncturing set in layer ℓ. Layer ℓ consists of 2^ℓ nodes that are either punctured or not. If we now map our 2^ℓ nodes to a $2^{\lceil \frac{\ell}{2} \rceil} \times 2^{\lfloor \frac{\ell}{2} \rfloor}$ board with coins on each cell, showing heads if the position is in punc and showing tails otherwise, then it is easy to see that we can use the same strategy to encode a message $\mathsf{msg} \in \{0,1\}^\ell$. The sender simply sets $\chi = \mathsf{msg}$ and then simulates Alice to modify punc accordingly.
The receiver can then act as Bob to retrieve the message msg. The receiver thus runs in polytime, as long as $2^\ell \in \mathsf{poly}(\lambda)$.