CryptoZoo: A Viewer for Reduction Proofs

Chris Brzuska, Christoph Egger, Kirthivaasan Puniamurthy

CryptoZoo: A Viewer for Reduction Proofs

Chris Brzuska[1], Christoph Egger[2], Kirthivaasan Puniamurthy[1]

[1] Aalto University, Finland
[2] Université Paris Cité, CNRS, IRIF, France

Abstract. Cryptographers rely on visualization to effectively communicate cryptographic constructions with one another. Visual frameworks such as constructive cryptography (TOSCA 2011), the joy of cryptography (online book) and state-separating proofs (SSPs, Asiacrypt 2018) are useful to communicate not only the construction, but also their *proof* visually by representing a cryptographic system as graphs.
One SSP core feature is the *re-use* of code, e.g., a package of code might be used in a game and be part of the description of a reduction as well. Thus, in a proof, the linear structure of a paper either requires the reader to turn pages to find definitions or writers to re-state them, thereby interrupting the visual flow of the game hops that are defined by a sequence of graphs.
We present an interactive proof viewer for state-separating proofs (SSPs) which addresses the limitations and perform three case studies: The equivalence between simulation-based and game-based notions for symmetric encryption, the security proof of the Goldreich-Goldwasser-Micali construction of a pseudorandom function from a pseudorandom generator, and Brzuska's and Oechsner's SSP formalization of the proof for Yao's garbling scheme.

Keywords: state-separation, proof viewer, reduction proofs, tooling

1 Introduction

Reduction proofs are a means to convince oneself and others of the security properties of a cryptographic design. In addition to communication and verification, (reduction) proofs help us gain understanding of the properties that are conducive to security and properties that are harmful. In order to improve verification, communication and understanding of proofs of complex systems, the cryptographic community has different techniques and styles which we briefly review below.

Black-box primitives. Black-box primitives are abstractions which support modular proofs and information-hiding. For example, the concept of a symmetric encryption scheme (SE) with indistinguishability under chosen plaintext attacks (IND-CPA) allows one to build a system which uses SE without delving (or even knowing about) the details of the AES cipher and suitable ciphermodes. Cryptographic proofs of a complex system typically use multiple such black-box

primitives. Additionally, a modular proof tends to first abstract away a sub-system, prove its security and then reduce the security of the bigger system to the sub-system in a black-box way. These neat black-box interfaces are a rich resource for structuring and understanding constructions and proofs, and they capture the *typical use* of primitives so that studying black-box proofs has become an established method also to understand the limitations of typical construction approaches [RTV04,BBF13].

Universal composability. Useful security definitions for *primitives* which compose well tend to consider *adversarially chosen* inputs, and Canetti's universal composition framework (UC [Can01]) applies this insight to *protocols* as well, thereby providing the basis to also prove *protocol* security by means of useful intermediate notions. Since UC specifies how the adversary and the (adversarial) environment can interact with the protocol, defining security in UC boils down to specifying an ideal functionality, without the need to re-invent (and possibly mis-construct) the adversarial model and enabling information-hiding since the UC-savvy reader can focus on reading the ideal functionality alone. Several further frameworks implement a similar approach, notably including Maurer's abstract cryptography framework [Mau11] and Rosulek's *Joy of Cryptography* [Ros21], which both encourage the use of visual components to follow proofs.

Game-hopping. Another important cryptographic technique that serves modular proofs are *game-hops*, explained by Shoup in [Sho04]. Game-hopping splits a large proof of indistinguishability between two games that formalize the security of a system into a sequence of smaller indistinguishability steps, each of which can be proven in isolation and the sequence of lemmas establishing the indistinguishability of each pairs of subsequent games then implies the indistinguishability of the two games in question.

Code-based game-hopping. Bellare and Rogaway [BR06] introduced the use of code into game-hopping proofs, where pseudo-code allows to put two subsequent games next to each other and inspect how exactly the code changes between them. Code-based game-playing has a strong visual component which draws the attention to the changes in a game-hop while at the same time keeping all the relevant code at hand for the reader (since it is written above and below the code which changes). In turn, game-hopping which is not code-based tends to describe only the changes from one game to another, requiring the reader to remember or recover the entire context from different parts of the paper (game definition, construction definition, previous game-hops).

Modular code-writing. In the realm of real-world protocols, code-based games are frequently used (see, e.g., [DDGJ22,DGGP21] for recent works) and/or code-based game-hopping is widely used (see, e.g., [DHRR22,DHK+23] for recent works). Interestingly, *reductions* are often specified in text or omitted—although some works diligently provide reductions in pseudo-code, e.g., the proof of Yao's garbling scheme by Bellare, Hoang and Rogaway (BHR [BHR12]). BHR employ

careful packaging of code into sub-routines in order to make the large amount of code in Yao's garbling scheme manageable and be able to argue that the reduction is sound. Concretely, reduction $\mathcal{R}$ that interacts with a smaller game $\text{Game}^b_{\text{small}}$ is sound if for the larger game $\text{Game}^b_{\text{big}}$ that the adversary $\mathcal{A}$ interacts with, it holds that for both $b \in \{0, 1\}$,

$$\mathcal{R} \to \text{Game}^b_{\text{small}} \equiv \text{Game}^b_{\text{big}}, \tag{1}$$

i.e., the behaviour of the reduction $\mathcal{R}$ composed with $\text{Game}^b_{\text{small}}$ is equivalent to the input-output behaviour of $\text{Game}^b_{\text{big}}$.

State-separating proofs. Modular code-writing with reductions in mind was systematically developed further by Brzuska, Delignat-Lavaud, Fournet, Kohbrok and Kohlweiss (BDFKK [BDF+18]) who structure the pseudo-code of a game into *stateful* pieces of code which can call one another, but else not access each others state whence the term *state-separation*. If $\text{Game}^b_{\text{big}}$ is defined by a call-graph of code packages and if the reduction $\mathcal{R}$ and the game $\text{Game}^b_{\text{small}}$ are defined via call-graphs of code packages as well, then we can compare the graph $\mathcal{R} \to \text{Game}^b_{\text{small}}$ obtained by "gluing" to the graph of $\text{Game}^b_{\text{big}}$. If the two call graphs are equal (i.e. they preserve the same edge relation), then Equation (1) holds trivially. Additionally, the reductions $\mathcal{R}$ can simply be defined by drawing a cut in the graph of $\text{Game}^b_{\text{big}}$, foregoing not only the need to prove the soundness of the reduction, but also the need to write the code of the reduction.

This *code re-use* approach is a core feature of SSPs which makes proofs concise and precise at the same time. On the other hand, code re-use also means that code packages are used many times across an article, while the package code is specified at a single place—or repeated redundantly at the cost of cluttering the paper and interrupting its flow. See [Koh23] for a gentle and thorough introduction to SSPs as well as a conceptual overview of recent works using and formalizing SSPs.

An SSP viewer. We address the limitations of paper-based presentation by designing the proof viewer *CryptoZoo*[3] for SSPs. CryptoZoo presents the claims and call-graphs of games in the left pane, and the code-based definitions in the right pane, the latter of which is available at all times, enabling code-linkage without code repetition, without breaking the flow and without the need to scroll away from a proof step in the current context. Additionally, clicking on individual packages highlights the relevant code in the code pane.

In addition to addressing these SSP-specific challenges, CryptoZoo also addresses presentational obstacles present for proofs in mathematics and information about systems in general and in cryptography specifically. We now briefly review three key areas where improvement of presentation is necessary and useful and then describe how CryptoZoo addresses these.

Information linkage Beyond code, proof steps reference different aspects, including previous lemmas and definitions. It is important to make this relationship functional and allow easy retrieval of all related facts.

[3] Available at `https://proofviewer.cryptozoo.eu`

Information hiding Since human memory is bounded, readers concentrate on facts and information immediately useful to the current task at hand. It is useful to *hide* information that does not contribute to understanding in a particular moment.

Soundness and structure Proofs are structured into claims and lemmas which form a proof tree (or DAG). The reader needs to verify that a set of claims indeed implies the statement of the parent node—and in the end, the reader can inspect the proofs of the claims at the leaves of the tree. In addition to the tree structure, the author of an article usually suggests a meaningful traversal which eases comprehension. A good medium for proofs should both provide the high-level tree structure and a recommended reading order while giving the reader freedom to deviate.

To address the latter point, CryptoZoo makes the proof tree visible and explicit at each point in the proof. In order to address retrieval speed challenges, CryptoZoo links cryptographic definitions and claims so that they can be retrieved quickly without losing the current context. Finally, to support user memory management and focus, CryptoZoo allows the user to hide text, definitions, lemmas, claims and their proofs, e.g., in order to focus on one particular subtree. The aforementioned approaches can be employed also generically when working with (cryptographic) proofs. It is, however, especially useful for the SSP method which inherently relies on a modular and visual approach. Additionally, SSPs have a quite well-defined set of proof steps which CryptoZoo supports while proofs in general (or even in cryptography) can be expressed in quite diverse ways which conflicts with the need of a proof viewing tool which supports more than the basic tree structure present in all mathematical proofs. For this reason, the SSP methodology is a useful scope for our proof-viewing tool.

Case studies. We provide three case studies for the SSP proof viewer. As a simple example, we show that the standard game-based notion of indistinguishability under chosen plaintext attacks (IND-CPA) and simulation-based security for symmetric encryption are equivalent (Section 5). A more advanced example is a state-separating proof of a (constant-depth) version of the Goldreich-Goldwasser-Micali (GGM) theorem which transforms pseudo-random generators (PRGs) into pseudorandom functions (PRFs) by using the PRG in a tree-structured construction. This proof is interesting, since it involves a two-dimensional hybrid argument, i.e., a hybrid argument both over the depth and width of the tree (Section 6). Finally, as an advanced example, we present a proof of Yao's garbling scheme in the version by Brzuska and Oechsner [BO23] (Section 7) which covers circuits which are structured in layers. This proof also involves a two-dimensional hybrid argument, both over the width and depth of the garbled circuit. In both proofs, SSPs allow to make the reductions explicit and visually accessible. The (constant-depth) GGM proof becomes visually straightforward using the proof viewer, although it is known as a rather complex proof in the foundations of cryptography. The rather involved proof of Yao's garbling scheme does not become straightforward, but its structure is significantly simplified—moreover, the proof viewing tools is useful due to the amount of code which needs to be managed.

Our case study on (constant-depth) GGM is the first formalization of the GGM proof in SSPs and useful to understand GGM—but it is also useful to understand SSPs, since it is the first intermediate-size SSP example. While BDFKK gave several simple examples, most follow-up work (e.g. [BCK22,BDE+22]) study complex protocols which are too complex to learn the SSP methodology based on them. The GGM proof is a nice middle-ground between simple and complex case studies.

Outline. We cover related work on visual tools in Section 2. Subsequently, Section 3 introduces SSPs and elaborate on the interrelation between useful aspects of SSPs and proof viewer design. We then discuss the proof viewer design and further considerations in the implementation (Section 4) and then turn to the three case studies.

2 Related Work

Visual aids are a natural match for teaching and can be found in teaching not only cryptography, but also computer science at large. For example, Vamonos [CR15] combines visual and (pseudo-)code aspects to communicate algorithmic correctness, while e.g. ProtoViz [Elm04] and GRACE [CDSP08] focus on the message flow in protocols. Crucial in these tools is the combination of exploration by the user together with a visual representation of the results. One can see editors for proof assistants like Coq [The17] and Lean [dMKA+15] in a similar spirit. The user provides a further proof step and is presented with the statements which still require a proof. Similarly, Tamarin [SMCB12]—a prover for protocol security—can display a graph of its internal reasoning and update it step-by-step.

While exploring a theorem statement in e.g. Coq can give insights and help students learn to formally reason, once proofs become complex this access becomes insufficient, in particular if proof search features are used. While exploring intermediate states of the proof remains helpful, the high level structure of a complex proof is not easily accessible from the linear proof file. Here tools like the prooftree [Tew] plugin for Coq can help to visually explore the dependencies between intermediate claims in a structured manner.

Alectryon [Pit20] adds a different dimension by allowing to freely switch between a textual proof description and the formally verified proof script (which can also be used to deduce the intermediate proof state). To this end it combines text formatting tools with the Coq prover (and has been extended to LEAN [Bül22]) to provide an interactive HTML document detailing the complete proof in a manner optimized to be digested by a human reader—while still guaranteed to be in sync with the mechanically verified version.

Visual Cryptographic Proofs The present work is inspired by the rational underlying SSPs, which bring the necessary rigor needed for formal verification while exposing similarities with teaching tools in cryptography which existed before, importantly, Rosulek's *Joy of Cryptography* [Ros21]. Rosulek groups blocks of code into libraries or packages and argues on the call-graph in a similar

(although less formal) way. This, in particular, allows students to draw from experience e.g. in object oriented programming where internal state of objects remain hidden and cannot be accessed. Maurer's dot-calculus in constructive cryptography [Mau11] facilitates proof communication by giving a visual outline on the relationship between objects.

3 State-Separating Proofs

The state-separating proofs (SSP) methodology by Brzuska, Delignat-Lavaud, Fournet, Kohbrok and Kohlweiss (BDFKK [BDF+18]) specifies not only a proof style for game-hopping proofs, but also a definitional style. Similar to the UC framework, SSP-style definitions specify security as indistinguishability between two games, typically a real and an ideal game. Indistinguishability is useful for composability, because even for (strong) unforgeability under chosen message attacks (UNF-CMA)—conceptually a search problem—a game-hop typically replaces real signature verification by log-based ideal signature verification, so that reductions between indistinguishability games tend to be more straightforward.

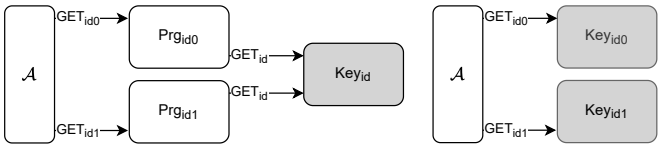

Fig. 1: Games Gprg^{0}_{id} and Gprg^{1}_{id}

Both in SSPs and UC, the adversary is the *main* algorithm which starts the system by activating other parts—in UC, the adversary activates the environment, the simulator or protocol parties (by sending messages to them), while in SSP-style games, the adversary activates the game by making oracle queries to it. For a game Game, we write $\Pr\left[1 = \mathcal{A}^{\mathsf{O}_1,\mathsf{O}_2} \text{ Game}\right]$ for the probability that adversary $\mathcal{A}$ returns 1 when interacting with the oracles O_1 and O_2 of Game, where the oracles O_1 and O_2 are defined via pseudo-code that operates on the state of the Game. An SSP-style game typically splits its code into multiple packages with separate state—a package, like a game, consists of a set of oracles operating on its state, but in the case of a package, oracles can make queries to the oracles of other packages as well, giving rise to a call-graph. As an example, consider a length-doubling pseudorandom generator (PRG) $g : \{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$ such that $g(x)$ (for x sampled uniformly at random) is indistinguishable from a uniformly random string y of length 2λ. We formalize security of PRGs as a game with two GET oracles, a GET_0 oracle which gives the adversary access to the left half of y and a GET_1 oracle which gives the adversary access to the right half of y. In the real game, the oracles return the left and right half of $y = g(x)$, respectively. In

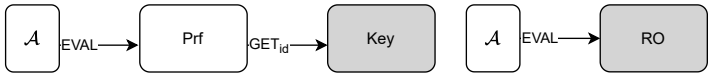

Fig. 4: The games Prf → Key and RO

the ideal game, the oracles both return a uniformly random string of length n. Modeling a PRG to return the chunks separately is equivalent to returning them at once, but will be useful in the security proof of the pseudorandom function (PRF) construction by Goldreich Goldwasser and Micali (GGM [GGM86]), where each half is post-processed separately.

We now define the ideal game as a *composition* of two smaller games Key_0 and Key_1 which we compose in parallel (cf. Figure 1 (right)). The GET_0 and GET_1 oracle of Key_0 and Key_1, respectively, each sample a uniformly random string and return it to the adversary. For the real game, we define two packages Prg_0 and Prg_1 whose oracles GET_0 and GET_1 each retrieve a key from the Key package via a GET oracle, then apply the PRG g to the value x they receive and return the left and right half of x, respectively.

$\underline{\underline{\mathrm{Key}_{id}}}$	$\underline{\underline{\mathrm{Prg}_{id0}}}$	$\underline{\underline{\mathrm{Prg}_{id1}}}$
Parameters	Parameters	Parameters
λ : sec. param	λ : sec. param	λ : sec. param
	g : PRG	g : PRG
State	State	State
x : string	no state	no state
GET_{id}	GET_{id0}	GET_{id1}
if $x = \bot$:	$x \leftarrow \mathsf{GET}_{id}()$	$x \leftarrow \mathsf{GET}_{id}()$
$x \leftarrow\!\!\$\ \{0,1\}^\lambda$	$z \leftarrow g(x)$	$z \leftarrow g(x)$
	$y \leftarrow z_{1..\frac{\lambda}{2}}$	$y \leftarrow z_{(\frac{\lambda}{2}+1)..\lambda}$
return x	**return** y	**return** y

Fig. 2: Code of Key_{id}, Prg_{id0} and Prg_{id1}

Definition 1 (Pseudorandom Generator). *A polynomial-time computable, deterministic function $g : \{0,1\}^* \to \{0,1\}^*$ with $\forall x \in \{0,1\}^*$, $|g(x)| = 2|x|$ is a PRG if for all indices $id \in \{0,1\}^*$ and all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, the following advantage is a negligible function in λ:*

$$\mathsf{Adv}(\mathcal{A}, \mathrm{Gprg}^0_{id}, \mathrm{Gprg}^1_{id}) := \left|\Pr\left[1 = \mathcal{A} \to \mathrm{Gprg}^0_{id}\right] - \Pr\left[1 = \mathcal{A} \to \mathrm{Gprg}^1_{id}\right]\right|.$$

Packages and adversaries receive the security parameter implicitly, i.e., advantage $\mathsf{Adv}(\mathcal{A}, \mathrm{Gprg}^0_{id}, \mathrm{Gprg}^1_{id})$ maps a value $\lambda \in \mathbb{N}$ to a number in the interval $[0,1]$.

Indices. Instead of defining the Key package in three variants, we simply allow it to carry a bitstring $id \in \{0,1\}^*$ as index and modifies oracle and package names for disambiguation, leading to Key, Key_0 and Key_1.

PRFs. As a 2nd example, we define a pseudorandom function. Here, we use the useful convention of defining the game *in terms of the construction*, i.e., we

define a pseudorandom function as a *stateless package* Prf. I.e., a pseudorandom function Prf is a package which (a) does not remember state between invocations, (b) makes use of a key from a Key package and (c) is indistinguishable from a random oracle, see Figure 4.

Definition 2 (Pseudorandom Function). *A pseudorandom function is a stateless package* Prf *which provides oracles* $[\to \text{Prf}] = \mathsf{EVAL}$ *and calls oracle* $[\text{Prf} \to] = \mathsf{GET}$ *such that for all PPT adversaries* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}(\mathcal{A}, \text{Prf} \to \text{Key}, \text{RO}) \quad := |\Pr[1 = \mathcal{A} \to \text{Prf} \to \text{Key}] - \Pr[1 = \mathcal{A} \to \text{RO}]|$$

is negligible in λ*.* Prf *and* RO *are assumed to use the same input length in.*

4 A proof viewer for SSPs

In Section 4.1, we discuss how we realize the proof viewing concepts outlined in Section 1 in the SSP proof viewer, and in Section 4.2, we consider further implementation considerations.

Prf Package	RO Package
Parameters	Parameters
λ : sec. param	λ : sec. param
in : input length	in : input length
State	State
no state	T : table
Oracle	$\mathsf{EVAL}(x)$
$[\to \text{Prf}]$: EVAL	**assert** $x \in \{0,1\}^{in}$
$[\text{Prf} \to]$: GET	**if** $T[x] = \bot$:
	$y \leftarrow\!\!\$\ \{0,1\}^n$
	$y \leftarrow T[x]$
	return y

Fig. 3: Code of Prf and RO

4.1 Proof Viewing Concepts

Linking and simultaneous visibility. CryptoZoo displays code of packages in a separate pane from games and lemmas, so that the reader can reach the code without losing the context of the games and lemmas they are currently studying. Additionally, clicking on a package will highlight the relevant code in the right pane (and scroll to it if needed). Additionally, CryptoZoo has clickable security definitions which open in a separate window.

It is possible to emulate those features partially in static PDF also via clickable packages, and opening several instances of the same PDF, which contain code and definitions, but achieving linking (showing and highlighting code when clicking) at the same time as simultaneous visibility would require non-standard links across several PDFs.

Proof structure and information hiding. When security definitions of a state separating proof are presented as suitable SSP graphs, a state-separating proof that involves many reductions can sometimes be not only more precise, but also shorter than a similar traditional proof since defining a reduction and proving its soundness consists only of drawing two graphs, cf. [BCK22]. In turn, when

proving equivalence with standard security definitions in addition, an SSP proof usually grows by at least two graphs and two inlining proofs for the high-level security definitions as well as at least two graphs and two inlining proofs for each of the underlying assumptions, cf. the proof of Yao's garbling scheme by Brzuska and Oechsner (BO [BO23]).

The proof viewer allows to hide sub-trees of a proof graph and, per default, hides code equivalence steps, but allows to display them next to the actual claim and relevant graphs and, again, with the code pane on the right. Again, one can emulate this feature partially by opening several instances of the same PDF, but the interactive hiding of arbitrary proof subtrees does not seem to be emulatable in PDF. An interesting and useful approach in static PDF has been taken in the thesis by Egger [Egg23] which presents security reductions for TLS 1.3 and first shows an overview proof tree and repeats sub-trees later in the relevant sections, recalling relevant context. Since CryptoZoo is not bound to linear structure, the user can fold and un-fold subtrees interactively. Additionally, the user can toggle between showing explanatory text or not, allowing to include comprehensive explanatory text while at the same allowing for compact representation.

4.2 Implementation Considerations

CryptoZoo is implemented as a web application, to allow user to access it without a dedicated application. To this end we believe assuming the availability of a web browser to be generally justified. The viewer is also designed to function offline, with minimal dependencies. Proofs/definitions are stored in a JSON format, which is loaded by the viewer when requested by the user.

5 Case study: IND-CPA vs. simulation-based security

Simulation-based security notions for symmetric encryption state that the adversary should not learn more than some ideal leakage and that everything the adversary can do when given a ciphertext can also be done when only given the ideal leakage, but not the ciphertext. While different views on ideal leakage are possible, the minimal approach is to leak the *length* of the message that is encrypted, since an adversary can infer the length of the message from the length of the ciphertext[4]. Simulation-based notions which leak the length of the message are typically equivalent to their game-based counterparts, see, e.g., [DF18].

In this case study[5], we provide an SSP-style proof showing that indistinguishability under chosen plaintext attacks (IND-CPA) security in its Real-or-Zeroes formulation is equivalent to a simulation-based formulation where the simulator receives only the length of the message m, encoded in unary as $0^{|m|}$. For completeness, let us state the correctness and security properties before turning to a discussion of the equivalence proof.

[4] Length-hiding encryption can mitigate this issue to some extent, but due to information-theory and correctness of decryption, the length of the ciphertext is always an upper bound on the length of the message.

[5] `https://proofviewer.cryptozoo.eu/sim-ind-cpa-landing.html`

Definition 3 (Symmetric Encryption Syntax). *A* symmetric encryption scheme se *consists of two probabilistic polynomial-time (PPT) algorithms* $\mathsf{se.Enc}$ *and* $\mathsf{se.Dec}$

$$c \leftarrow\!\!\$\, \mathsf{se.Enc}(k, m)$$
$$m \leftarrow \mathsf{se.Dec}(k, c)$$

which satisfy that for all security parameters $n \in \mathbb{N}$*, encryption is correct, i.e.,*

$$\forall m \in \{0,1\}^* \Pr_{k \leftarrow\!\!\$ \{0,1\}^n}[\mathsf{se.Dec}(k, \mathsf{se.Enc}(k, m)) = m] = 1.$$

Definition 4 (IND-CPA security). *A symmetric-encryption scheme* se *is* indistinguishable under chosen plaintext attacks *if for all PPT adversaries* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}(\mathcal{A}, \mathrm{Genc}^0, \mathrm{Genc}^1) := \left|\Pr\left[1 = \mathcal{A} \to \mathrm{Genc}^0\right] - \Pr\left[1 = \mathcal{A} \to \mathrm{Genc}^1\right]\right|$$

is a negligible function in the security parameter, where $\mathrm{Genc}^0 := \mathrm{Enc} \to \mathrm{Key}$ *and* $\mathrm{Genc}^1 := \mathrm{Zeroer} \to \mathrm{Enc} \to \mathrm{Key}$*,* Key *is defined in Figure 2, and* Zeroer *and* Enc *are defined in Figure 7.*

Definition 5 (Simulation-based security). *A symmetric-encryption scheme* se *satisfies* simulation-based security *if there exists a PPT simulator* Sim *such that for all PPT adversaries* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}(\mathcal{A}, \mathrm{Genc}^0, \mathrm{Genc}(\mathsf{Sim})) := \left|\Pr\left[1 = \mathcal{A} \to \mathrm{Genc}^0\right] - \Pr[1 = \mathcal{A} \to \mathrm{Genc}(\mathsf{Sim})]\right|$$

is a negligible function in the security parameter, where $\mathrm{Genc}^0 := \mathrm{Enc} \to \mathrm{Key}$ *and* $\mathrm{Genc}(\mathsf{Sim}) := \mathrm{Zeroer} \to \mathrm{Sim}$ *are defined in Figure 7.*

We will see that the ideal encryption game can act as a simulator, since all the simulator needs to do is to encrypt zeroes. In this way, we obtain a straightforward proof that IND-CPA security implies the simulation-based security notion for symmetric encryption (See Figure 5). In the converse direction, we need to use the simulation-based security notion twice in the proof—once to move away from encrypting real messages to encrypting simulated messages, and once to argue that encrypting simulated messages is indistinguishable from encrypting zeroes—since the ideal functionality which gives $0^{|m|}$ to the simulator yields

Enc^0 Package	Zeroer Package
Parameters	Parameters
λ : sec. param	no parameters
se : sym. enc. sch.	
State	State
no state	no state
$\mathsf{ENC}(m)$	$\mathsf{ENC}(m)$
$k \leftarrow \mathsf{GET}()$	$m' \leftarrow 0^{\lvert m\rvert}$
$c \leftarrow\!\!\$\ \mathsf{se.enc}(k, m)$	$c \leftarrow \mathsf{ENC}(m')$
return c	**return** c

Fig. 7: Package definitions for IND-CPA and simulation-based security of se.

Claim 1: Equivalence

$$\mathrm{Genc}^1 \overset{code}{\equiv} \mathrm{Genc}(\mathrm{Sim}_{Lemma2})$$

Proof of Claim 1

Below, in $\mathrm{Genc}(\mathrm{Sim}_{\mathrm{Lemma2}})$, we replace Sim_{Lemma2} by its definition and thereby obtain $\mathrm{Hybrid\text{-}Claim\text{-}1}$. We mark the definition of $\mathrm{Genc}(\mathrm{Sim}_{\mathrm{Lemma2}})$ by a dashed line, Removing the dashed line yields Genc^1.

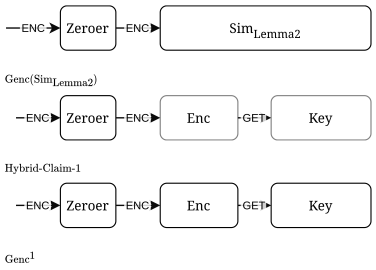

Fig. 5: Using the real game as a simulator.

Proof of Lemma 1

Let $\mathcal{A}$ be a PPT adversary. We prove Lemma 1 via three game-hops and bound the advantage of $\mathcal{A}$ via the triangle inequality.

$$\begin{aligned}\mathsf{Adv}(\mathcal{A}, \mathrm{Genc}^1, \mathrm{Genc}^0) \leq\ & \mathsf{Adv}(\mathcal{A}, \mathrm{Genc}^1, \mathrm{Hybrid\text{-}Lemma1}) \\ & + \mathsf{Adv}(\mathcal{A}, \mathrm{Hybrid\text{-}Lemma1}, \mathrm{Genc}(\mathsf{Sim})) \\ & + \mathsf{Adv}(\mathcal{A}, \mathrm{Genc}(\mathrm{Sim}), \mathrm{Genc}^0)\end{aligned}$$

Where the first and last advantage are obtained by reduction to simulation security while the middle term follows from code equivalence and thus is 0. The proof is visualized by the following sequence of games:

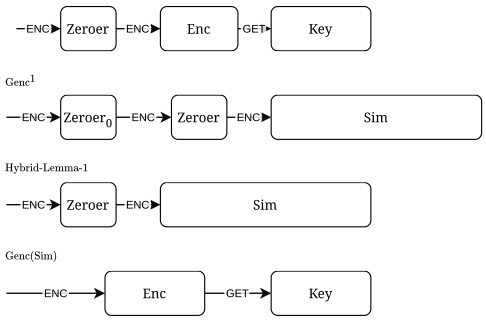

Fig. 6: Using the real game as a simulator.

the same output, regardless of whether m is an all-zeroes string or not. These arguments become visible in the proof structure (see Figure 6) and its associated proof graphs. The game hop from Genc^1 to Hybrid-Lemma-1 is a reduction step which can be visualized as a cut in a graph, depicted in Figure 8 which hatches the reduction in red. The last game hop is directly implied by the indistinguishability of the real game Genc^0 and the simulated game Genc(Sim), and the middle game hop follows by observing that two Zeroer packages are equivalent to a single one.

Claim 2: Indistinguishability between **Genc** and **Hybrid-Lemma-1**

For all simulators Sim and all adversary $\mathcal{A}$,

$$\mathsf{Adv}(\mathcal{A}, \text{Genc}^1, \text{Hybrid-Lemma1}) = \mathsf{Adv}(\mathcal{A} \to \text{Zeroer}, \text{Genc}^0, \text{Genc}(\text{Sim}))$$

Proof of Claim 2

Using Zeroer as a reduction, we observe that the remaining part of the graph constitute exactly Genc^0 and $\text{Genc}(\text{Sim})$, respectively, and Claim 2 follows.

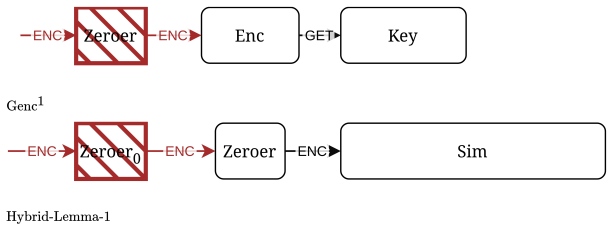

Fig. 8: Reduction hatched in red.

6 Case study: Constant-depth GGM tree

Goldreich, Goldwasser and Micali (GGM [GGM86]) introduced the notion of pseudorandom functions and provided a construction of a pseudorandom function based on a length-doubling pseudorandom generator (see Definition 1). The proof is commonly include in courses on the foundations of cryptography and contained, e.g., in Chapter 3.6.2 (Theorem 3.6.6) of the *Foundations of Cryptography I* textbook by Goldreich [Gol04]. The construction is naturally amenable to visualization: It structures PRG instances into a binary tree and the left halves and right halves of a PRG output become the input to the PRG instances on the next tree layer, until reaching the leaf layer.

The construction[6] is indeed often visualized as a tree. See, e.g., Figure 3.5 in Chapter 3.6.2 of [Gol04]. As we will see, not only the construction, but also the proof can be visualized. We will see how the security of each of the PRG instances is applied and how the reduction looks like. The proof is a hybrid argument over all of the PRG instances. For illustration, consider the two hybrid games in Figure 9. Their difference can be reduced to the PRG security by using the boxed hatched in red as a reduction (Figure 10).

The proof which we have chosen to implement into the proof viewer is a variant of the GGM proof where the tree is of constant depth. This is analogous to how the GGM construction is often depicted in books, namely restricted to a constant level, since the full GGM construction is an n-level tree with 2^n leaves which is harder to represent than a finite tree.

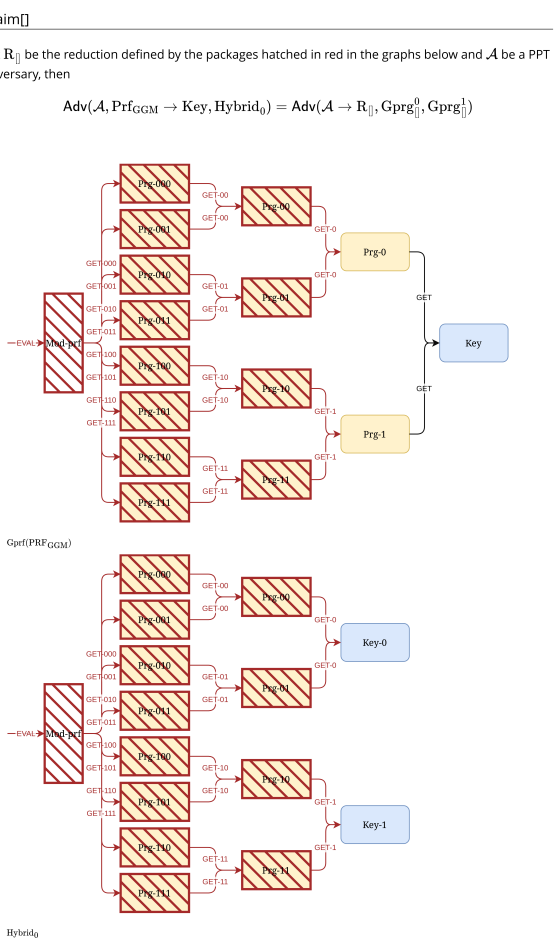

Fig. 9: Hybrid step in the GGM proof

We depict the hybrid argument for tree depth 3 which requires $2^3 - 1 = 7$ PRG instances and thus also 7 game-hops. Each of the hybrid games is represented via a binary tree. The format of the proof viewer is convenient here since we avoid page boundaries and can depict the hybrids simply as a long sequence of 8 games. Note that the full GGM proof does not proceed via a hybrid over the entire tree, but only visits polynomially many of the PRG instances in the tree. Our constant-depth representation does not capture this subtlety of the proof, and the SSP version of the full GGM proof that we are aware of, is visually not as appealing (see Section 7 for a compelling hybrid argument over polynomially many hybrids). Therefore, we prefer to present a constant-depth version of GGM which captures the main essence of the construction and, importantly, its security proof.

[6] `https://proofviewer.cryptozoo.eu/ggm-landing.html`

Claim0

Let R_0 be the reduction defined by the packages hatched in red in the graphs below and $\mathcal{A}$ be a PPT adversary, then

$$\mathsf{Adv}(\mathcal{A}, \mathrm{Hybrid}_0, \mathrm{Hybrid}_1) = \mathsf{Adv}(\mathcal{A} \to \mathrm{R}_0, \mathrm{Gprg}_0^0, \mathrm{Gprg}_0^1)$$

.

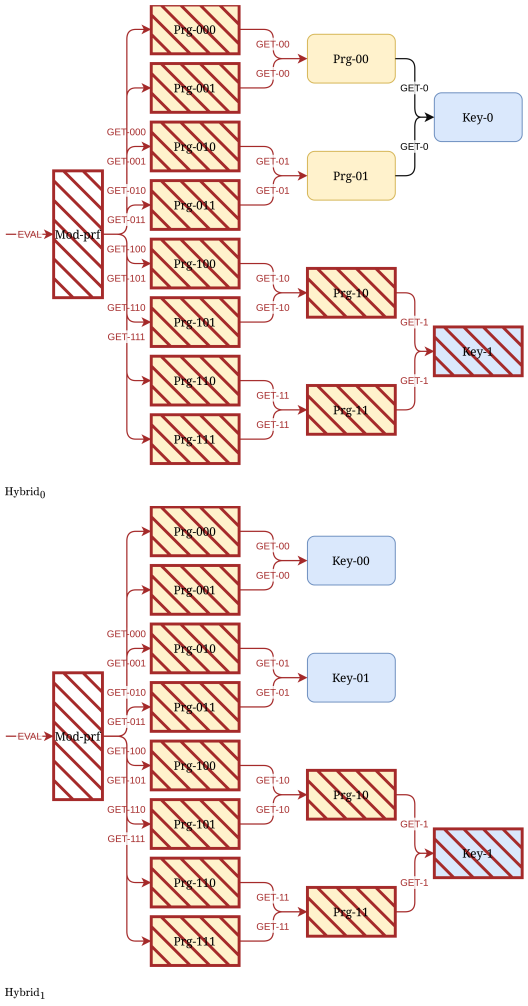

Fig. 10: Reduction step in the GGM proof

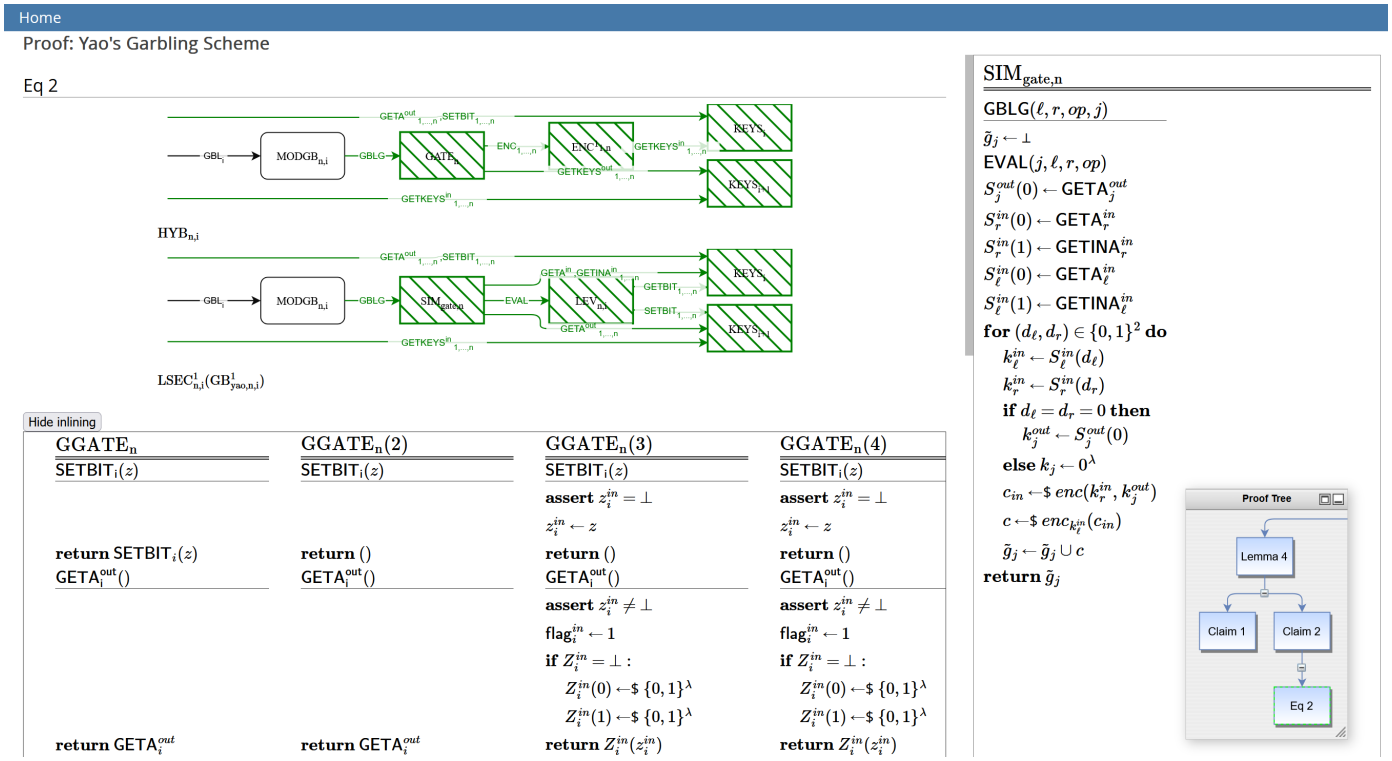

Fig. 11: Semantic switch step in the proof viewer (Eq. 2 of Yao, proof via inlining).

Fig. 12: Hybrids code equivalence in the proof viewer (Eq. 7 of Yao, proof via graph equality).

7 Case Study: Yao's Garbling Scheme

Secure multi-party computation constructs protocols where several parties *together* compute a function on the participants' input but without revealing their inputs to other protocol participants (beyond what the output of the function leaks). Yao [Yao86] proposed a protocol for this purpose which we know today as *garbled circuits*. For analyzing the security of this construction, BHR [BHR12] extracted the intermediate notion of a garbling scheme and proved security of Yao's garbling scheme. Brzuska and Oechsner [BO23] give a state-separated proof[7] of garbling security for *layered* circuits where they further extract a notion of layer garbling which allows sequential composition and therefore allows structuring the security proof by composing security layer-by-layer. This step is inherently visual (Figure 12). Moreover, to define these reductions for the viewer, the proof author need not produce separate graphs and code, but only needs to specify cuts on the same graph, enabling code re-use.

On a gate-by-gate level, Yao's garbling scheme operates by assigning two SE keys to each wire in a circuit representing the values 0 and 1. A binary logic gate can then be implemented by encrypting the key on the output wire under both input wire keys for all entries of the truth table: for a logic AND gate the output 1-key is encrypted under the 1-keys for both the left and right input wire while the output 0-key is encrypted under all other combinations of the input keys. Consequently, a party can access the output 1-key exactly if it knows both 1-keys for the input wires whereas the 0-key remains hidden in this case.

A central element of the security proof is a semantic switch – instead of considering 0-keys and 1-keys the security game distinguishes between "active" keys known to the party and "passive" keys which remain secret which makes the party's state independent of the actual input *value* when evaluating the circuit (Figure 11). The formal code equivalence proof of this step is quite tedious, but the proof viewer keeps components necessary to verify this step close together, which should help the reader.

8 Comparison.

We now review SSP proofs for the Transport Layer Security (TLS [Res18]) protocol, the Messaging Layer Security (MLS [BMO+23]) protocol and Yao's garbling scheme. In each case, we discuss the presentation of the respective papers and how CryptoZoo could further contribute to communication, and, in the case of Yao's garbling scheme, how CryptoZoo compares to the original presentation by Brzuska and Oechsner [BO23]. Afterwards, we briefly discuss SSP proofs in formal verification tools.

8.1 Yao's garbling scheme

Brzuska and Oechsner (BO [BO23]) formalize security and correctness of garbling schemes in SSPs and then revisit Yao's garbling scheme. While correctness can also

[7] `https://proofviewer.cryptozoo.eu/yao-landing.html`

be proven in SSPs, BO focus on security only and provide an SSP-style reduction for Yao's garbling scheme to the IND-CPA security of the underlying garbling scheme. The BO paper introduces games for IND-CPA security and garbling security in a natural flow, slowly increasing complexity in order to familiarize the reader with the novel SSP encoding and, as a first proof, show an equivalence for different encodings for IND-CPA security. Our CryptoZoo implementation follows this outline by BO. Concretely, the CryptoZoo landing page for Yao's garbling scheme explains the purpose of the different pages and then recommends to the reader to first visit the IND-CPA security page which explains the SSP encoding of IND-CPA security and the equivalence proof with the encoding of IND-CPA security which is useful for the security proof of Yao's garbling scheme. Subsequently, BO discuss the SSP encoding of garbling scheme security and Yao's garbling scheme construction. Our CryptoZoo implementation follows this approach and provides a page introducing the garbling scheme security notion in SSP-style and also explains Yao's garbling scheme construction. Thus, up to the main theorem statement, the BO paper and our CryptoZoo implementation proceed analogously.

The main difference between CryptoZoo and the BO presentation is the proof of the main theorem which reduces security of Yao's garbling scheme to IND-CPA security. BO proceed in a bottom-up fashion, slowly building and explaining sub-packages needed in the proof and showing equivalence with the top-level security notion in the end.

In turn, CryptoZoo natively presents the proof in a top-down fashion and explains the code of the modular packages previously in the context of the Yao construction. Below the statement of the main theorem, CryptoZoo recommends to the reader, however, to first read the proof bottom-up and then, once more, top-down. CryptoZoo allows the reader to proceed through the proof in both directions, since clicking on a lemma hides all the remaining proof steps, focusing solely on the lemma and its sub-tree. The reason that we first recommend a bottom-up reading of the proof is analogous to the presentation rationale of BO: The reader's familiarity with all packages grows successively with each proof step until reaching a statement for the entire garbling construction. In turn, reading the proof top-down in the first reading iteration either requires reading and understanding all code at once or treating some of the packages as black-boxes (since most of the proof steps are purely syntactical). However, after a first bottom-up read that helps familiarizing with all code and steps, making a top-down pass through the proof seems useful to gain a conceptual understanding of how the proof connects the high-level garbling security notion to the low-level IND-CPA definition. CryptoZoo supports both, the bottom-up and the top-down reading flow, and the user can, of course, also read the proof in an arbitrary order based on their preference. The CryptoZoo proof tree and information-hiding helps the user to engage with the proof conveniently in an order of their choice while having all information conveniently at hand. In turn, the BO proof has a fixed order where code has a fixed place in the paper and needs to be manually connected. As mentioned previously, opening multiple PDFs of BO (and adding

a proof tree to their paper) will reach a similar effect, but at a lower level of convenience than in CryptoZoo.

8.2 SSP proofs of TLS 1.3

Brzuska, Delignat-Lavaud, Egger, Fournet, Kohbrok and Kohlweiss (BDEFKK [BDE+22]) analyze the TLS 1.3 key schedule and Egger [Egg23] further connects the TLS 1.3 key schedule security with the TLS 1.3 handshake security. BDEFKK and Egger both introduce different code, assumptions and games in a natural flow, starting from a conceptually simple game (collision-resistance of hash-functions in BDEFKK and PRF security in Egger).

A remarkable property of the TLS 1.3 security analysis is the strongly layered approach: Each layer comes with a main theorem which builds upon the result of the previous layer as well as additional lemmata specific to the current layer. As such each layer in isolation can be an insightful read, e.g. to learn how to relate key schedule hand handshake security. Highlighting such additional ad-hoc structure of the proof tree is easy to do in a PDF presentation, and Egger's thesis follows this approach with a proof tree with clickable lemma statements and chapters zooming in (both visually and content wise) into each layer of the tree. While currently not implemented, adding this layer structure to CryptoZoo would be a reasonable task if it turns out to be applicable to many projects. Finally, both, for BDEFKK and Egger (as well as in a possible future CryptoZoo implementation), the proof trees are also useful to compute final advantage statements, see [Egg23, p.48,p.56].

8.3 SSP proofs of the MLS key schedule

Brzuska, Cornelissen and Kohbrok (BCK [BCK22]) analyze the MLS key schedule and its composition with TreeKEM [BBR18]. Again, BCK slowly build up complexity in their article and a CryptoZoo implementation would proceed analogously. Again, the main advantage of CryptoZoo lies in the availability and easy accessiblity of code, and in this case, also in an additional proof tree—but a proof tree could also be added to BCK. Again, a proof tree would be useful to compute final advantage statements, cf. [BCK22, p.18-19].

8.4 Formal verification tools for SSPs

SSProve is a Coq-based formal verification tool for SSPs by Abate, Haselwarter, Rivas, Van Muylder, Winterhalter, Hritcu, Maillard and Spitters [AHR+21], and Dupressoir, Kohbrok and Oechsner [DKO22] formalized SSPs in EasyCrypt [BDG+14,BGHZ11]. Representation of SSPs in both, SSProve and EasyCrypt, is code-based and thus, CryptoZoo could help present the obtained proof visually. Potentially, CryptoZoo code could be generated automatically and thus not only help in proof communication but perhaps also in proof development, allowing the proof developer faster visual navigation of the proof draft.

9 Conclusion and future work

One useful feature of visual(izable) frameworks such as UC, abstract cryptography, the Joy of Cryptography and SSPs is the visualization of *proofs*. In this article, we explored the presentation of SSP proofs in the *interactive* proof viewer CryptoZoo which we developed. We would like to claim that CryptoZoo improves the *quality* of verification by providing improved navigation of proofs and by allowing users to conveniently and quickly retrieve relevant information. However, readers of PDFs can compensate by retrieving information in different (slower) ways (cf. Section 4.1). Therefore, it seems more accurate that CryptoZoo improves the *speed* of verification or the quality of verification given a fixed, limited amount of time.

Future Work. It would be interesting to conduct a user study to compare the verification of (well-written) PDF proofs with the verification of (well-written) CryptoZoo proofs. Furthermore, it would be interesting to see whether CryptoZoo is useful for helping a proof developer maintain state in a visual form while writing an SSP proof. Last, but not least, CryptoZoo might be connected with formal verification tools for SSPs, such as SSProve [AHR+21] and or a formalization of SSPs in EasyCrypt [BDG+14,BGHZ11]. In this case, reduction steps and, more importantly, code-equivalence steps could be verified by in the underlying tool (rather than by the user/reader), turning CryptoZoo into an interface which helps a user/reader gain understanding of proof conducted in a formal verification tool and thus serve to ease a notoriously hard communication task.

Acknowledgments This project was supported by the Research Council of Finland and the European Commission under the Horizon2020 research and innovation programme, Marie Sklodowska-Curie grant agreement No 101034255.

References

AHR+21. Carmine Abate, Philipp G. Haselwarter, Exequiel Rivas, Antoine Van Muylder, Théo Winterhalter, Catalin Hritcu, Kenji Maillard, and Bas Spitters. SSProve: A foundational framework for modular cryptographic proofs in coq. In Ralf Küsters and Dave Naumann, editors, *CSF 2021 Computer Security Foundations Symposium*, pages 1–15. IEEE Computer Society Press, 2021.

BBF13. Paul Baecher, Christina Brzuska, and Marc Fischlin. Notions of black-box reductions, revisited. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 296–315. Springer, Heidelberg, December 2013.

BBR18. Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS). Research report, Inria Paris, May 2018.

BCK22. Chris Brzuska, Eric Cornelissen, and Konrad Kohbrok. Security analysis of the MLS key derivation. In *2022 IEEE Symposium on Security and Privacy*, pages 2535–2553. IEEE Computer Society Press, May 2022.

BDE+22. Chris Brzuska, Antoine Delignat-Lavaud, Christoph Egger, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. Key-schedule security for the TLS 1.3 standard. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 621–650. Springer, Heidelberg, December 2022.

BDF+18. Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. State separation for code-based game-playing proofs. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 222–249. Springer, Heidelberg, December 2018.

BDG+14. Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2014.

BGHZ11. Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 71–90. Springer, Heidelberg, August 2011.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

BMO+23. R. Barnes, J. Millican, E. Omara, K. Cohn-Gordon, and R. Robert. The Messaging Layer Security (MLS) Protocol. RFC 9420, 2023.

BO23. C. Brzuska and S. Oechsner. A state-separating proof for yao's garbling scheme. In *2023 2023 IEEE 36th Computer Security Foundations Symposium (CSF) (CSF)*, pages 127–142, Los Alamitos, CA, USA, jul 2023. IEEE Computer Society.

BR06. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

Bül22. Niklas Bülow. Proof visualization for the lean 4 theorem prover, April 2022.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CDSP08. Giuseppe Cattaneo, Alfredo De Santis, and U Ferraro Petrillo. Visualization of cryptographic protocols with grace. *Journal of Visual Languages & Computing*, 19(2):258–290, 2008.

CR15. B. Carmer and M. Rosulek. Vamonos: Embeddable visualizations of advanced algorithms. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, 2015.

DDGJ22. Hannah Davis, Denis Diemert, Felix Günther, and Tibor Jager. On the concrete security of TLS 1.3 PSK mode. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 876–906. Springer, Heidelberg, May / June 2022.

DF18. Jean Paul Degabriele and Marc Fischlin. Simulatable channels: Extended security that is universally composable and easier to prove. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 519–550. Springer, Heidelberg, December 2018.

DGGP21. Jean Paul Degabriele, Jérôme Govinden, Felix Günther, and Kenneth G. Paterson. The security of ChaCha20-Poly1305 in the multi-user setting. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1981–2003. ACM Press, November 2021.

DHK+23. Julien Duman, Kathrin Hövelmanns, Eike Kiltz, Vadim Lyubashevsky, Gregor Seiler, and Dominique Unruh. A thorough treatment of highly-efficient NTRU instantiations. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 65–94. Springer, Heidelberg, May 2023.

DHRR22. Benjamin Dowling, Eduard Hauck, Doreen Riepel, and Paul Rösler. Strongly anonymous ratcheted key exchange. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 119–150. Springer, Heidelberg, December 2022.

DKO22. François Dupressoir, Konrad Kohbrok, and Sabine Oechsner. Bringing state-separating proofs to EasyCrypt A security proof for cryptobox. In *CSF 2022 Computer Security Foundations Symposium*, pages 227–242. IEEE Computer Society Press, August 2022.

dMKA+15. Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

Egg23. Christoph Egger. *On Abstraction and Modularization in Protocol Analysis*. Doctoral thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2023.

Elm04. Niklas Elmqvist. Protoviz: A simple security protocol visualization. *the University of Gothenburg, Tech. Rep*, 2004.

GGM86. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

Gol04. Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

Koh23. Konrad Kohbrok. *State-Separating Proofs and Their Applications*. Doctoral thesis, Aalto University School of Science, 2023.

Mau11. Ueli Maurer. Constructive cryptography–a new paradigm for security definitions and proofs. In *Joint Workshop on Theory of Security and Applications*, pages 33–56. Springer, 2011.

Pit20. Clément Pit-Claudel. Untangling mechanized proofs. In Ralf Lämmel, Laurence Tratt, and Juan de Lara, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2020, Virtual Event, USA, November 16-17, 2020*, pages 155–174. ACM, 2020.

Res18. Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

Ros21. Mike Rosulek. *The joy of cryptography*. Oregon State University, 2021. Draft of January 3, 2021.

RTV04. Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 1–20. Springer, Heidelberg, February 2004.

Sho04. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `https://eprint.iacr.org/2004/332`.

SMCB12. Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In Steve Zdancewic and Véronique Cortier, editors, *CSF 2012 Computer Security Foundations Symposium*, pages 78–94. IEEE Computer Society Press, 2012.

Tew. Hendrik Tews. Prooftree.

The17. The Coq Development Team. The coq proof assistant, version 8.7.0, October 2017.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.