The proceedings version of this paper appears at CRYPTO '19. This is the full version.

Security in the Presence of Key Reuse: Context-Separable Interfaces and their Applications

Christopher Patton and Thomas Shrimpton

Florida Institute for Cybersecurity Research
Computer and Information Science and Engineering
University of Florida
`{cjpatton,teshrim}@ufl.edu`

Abstract

Key separation is often difficult to enforce in practice. While key reuse can be catastrophic for security, we know of a number of cryptographic schemes for which it is provably safe. But existing formal models, such as the notions of joint security (Haber-Pinkas, CCS '01) and agility (Acar et al., EUROCRYPT '10), do not address the full range of key-reuse attacks—in particular, those that break the abstraction of the scheme, or exploit protocol interactions at a higher level of abstraction. This work attends to these vectors by focusing on two key elements: the *game* that codifies the scheme under attack, as well as its intended adversarial model; and the underlying *interface* that exposes secret key operations for use by the game. Our main security experiment considers the implications of using an interface (in practice, the API of a software library or a hardware platform such as TPM) to realize the scheme specified by the game when the interface is shared with other unspecified, insecure, or even malicious applications. After building up a definitional framework, we apply it to the analysis of two real-world schemes: the EdDSA signature algorithm and the Noise protocol framework. Both provide some degree of *context separability*, a design pattern for interfaces and their applications that aids in the deployment of secure protocols.

Keywords: Key reuse, APIs, Diffie-Hellman, EdDSA, Noise

Contents

1 Introduction **3**

2 Pseudocode and Conventions **7**

3 Interfaces and Games **8**

4 Security Under Exposed Interface Attack **9**
- 4.1 Simulatability of an Interface . . . 11
- 4.2 The Composition Theorem . . . 12

5 Discrete Log Interfaces **14**
- 5.1 Diffie-Hellman . . . 15
- 5.2 EdDSA . . . 17

6 Noise **19**
- 6.1 Handshake and message patterns . . . 20
- 6.2 The interface . . . 22
- 6.3 Security . . . 24
- 6.4 Composition with EdDSA . . . 25
- 6.5 Conclusion . . . 26

A Proofs **31**
- A.1 Theorem 1 (composition) . . . 31
- A.2 Theorem 2 (necessity of wGAP2 for Theorem 1(ii)) . . . 33
- A.3 Theorem 4 (wGAP2 insecurity of functional DH) . . . 34
- A.4 Theorem 5 (GAP1 security of EdDSA) . . . 36
- A.5 Theorem 6 (GAP2 security of EdDSA) . . . 38
- A.6 Theorem 7 (GAP1 security of Noise) . . . 39
- A.7 Sketch of Theorem 8 (GAP2 security of Noise) . . . 44

B Indifferentiability of HKDF **44**

1 Introduction

The principle of *key separation*, or ensuring that distinct cryptographic functionalities use distinct keys, is a widely accepted tenet of applied cryptography. It appears to be difficult to follow, however, as there are many instances of *key reuse* in deployed cryptosystems, some having significant impact on the security of applications. The problem is multi-faceted. First, operational requirements of the system often demand some degree of key reuse. For example, it is common to use a signing key deployed for TLS [54] in other protocols, as this is permitted by certificate authorities and avoids the cost of certifying a distinct key for each protocol. But doing so has side effects that must be addressed in the design of these protocols, as well as the interface that exposes the key [15]. Second, it is often not clear what constitutes a "distinct functionality". Intel's Trusted Platform Module (TPM) standard [60] supports a variety of protocols for remote attestation that use an Intel-certified key stored on chip. The TPM exposes a core set of operations involving this key via its application-programming interface (API), which applications make calls to in order to implement attestation schemes. But the requirement to support so many protocols has lead to a flexibile API with subtle vulnerabilities [3, 20].

Prior work sheds light on when key reuse is safe among specific primitives. Haber and Pinkas [34] introduce the notion of *joint* security, which captures the security of a target cryptosystem (say, a digital signature scheme) in the presence of an oracle that exposes a related secret-key operation (say, the decryption operation of a public-key encryption scheme). Many widely used primitives are jointly secure, including RSA-PSS/OAEP [34] and Schnorr signatures/hybrid encryption [22]. Acar et al. [2] address the related problem of *agility*, where the goal is to identify multiple instantiations of a particular primitive (e.g., sets of AEAD schemes, PRFs, or signature schemes) that can securely use the same key material. But the range of potential key-reuse attacks goes well beyond what these works cover; attack vectors sometimes break the intended abstraction boundary of the scheme by exposing lower-level operations [19, 3], or involve unforeseen protocol interactions at a higher level of abstraction [36, 15]. We believe that a comprehensive treatment of key reuse can and should account for these attack vectors as well.

To this end, we propose to surface the API as a first class security object. For our purposes, the API (or just "interface") is the component of a system that exposes to applications a fixed set of operations involving one or more secret keys. APIs are often the root-of-trust of applications: TPM, Intel's Software Guard Extensions (SGX), hardware security modules (HSMs), and even chip-and-pin credit cards all provide cryptographic APIs that aim to be trustworthy-by-design. But pressure to meet operational requirements, while exporting interfaces that are suitable for a variety of applications, often leads to vulnerabilities [17, 22, 41, 3]. An analogous situation arises in the development of software that uses a cryptographic library; software engineers tend to trust that any use case *permitted* by an API is secure, without fully grasping its side-effects [50]. This trust tends to lead to vulnerable code [4, 51].

In light of these issues, this work seeks to develop security-oriented design principles for interfaces and their applications. We devise a definitional framework for reasoning about the security of an application when the interface it consumes is used in other, perhaps unintended or even insecure ways. We model these "other applications" very conservatively, as follows: to assist it in its attack against the target application, we assume the adversary has *direct* access to the underlying interface, allowing it to mount *exposed interface attacks* on a target application. We apply this framework to the design and analysis of two real-world cryptosystems: the EdDSA signature algorithm [35] and the Noise protocol framework [52]. In doing so, we elicit a property of interfaces and their applications we call *context separability*, which we will show to be an invaluable tool for secure protocol design.

The framework. We begin by motivating our definitional viewpoint, which draws abstraction boundaries a bit differently than usual. Game-based notions of security [12] typically specify (in pseudocode) a game $\mathcal{G}$ that makes calls to a cryptographic scheme Π (a primitive or protocol, also specified in pseudocode). The game captures an attack model—that is, the capabilities and goal of the adversary—and establishes boundaries on the permitted uses of Π. Model-specific adversarial capabilities are captured as oracle procedures specified by $\mathcal{G}$, which the adversary may query during its attack. Its goal is formalized by an explicit winning condition that depends on its queries and the random choices of the game. The security of the scheme, when used as specified by $\mathcal{G}$, is measured by executing an adversary with $\mathcal{G}$.

Suppose that Π is specified in terms of calls to an underlying interface $\mathcal{I}$, which defines the set of operations that can be performed on the secret key. Our goal is to measure the security of Π in the sense

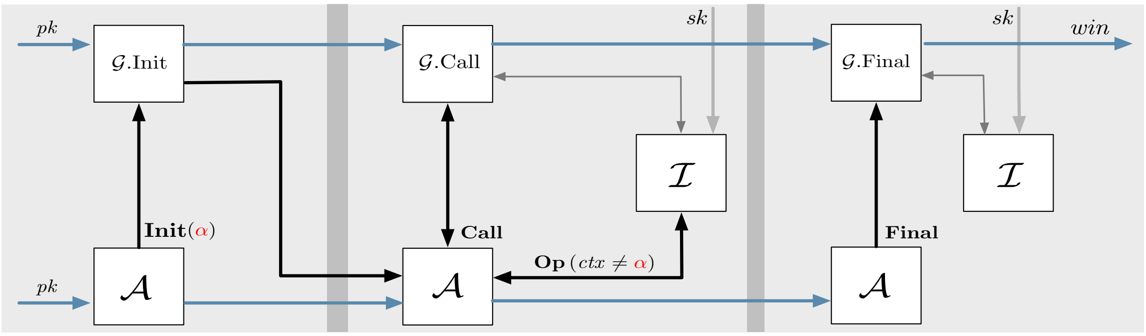

Figure 1: Illustration of the SEC/I experiment, which has three "phases": first, the adversary $\mathcal{A}$ chooses the game context α and initializes the game $\mathcal{G}$; second, $\mathcal{A}$ plays $\mathcal{G}$ and interacts with $\mathcal{I}$; and third, $\mathcal{A}$ finalizes $\mathcal{G}$ and the experiment outputs the outcome *win*.

---

of $\mathcal{G}$ when the adversary playing the game is also provided direct access to $\mathcal{I}$, i.e., when the adversary is able to mount exposed interface attacks on the security of Π that $\mathcal{G}$ codifies.

We formalize our syntax for interfaces and games in Section 3. Rather than refer explicitly to Π, we allow the game $\mathcal{G}$ to realize Π as pseudocode that makes calls to $\mathcal{I}$. Interfaces may expose conventional primitive operations like signing or decryption, or they may expose lower level operations that are composed into higher level ones by the game. (This is precisely what TPM does; more on this in Section 5.1.) Our syntax for interfaces admits operations on symmetric and asymmetric keys. In the latter case, all secret-key operations are handled by the interface, and all public-key operations are specified by the game.

Security under exposed interface attack. The objects of our study are an interface and a target application; we formalize the latter as a game that defines the scheme, how it is used, and what is its goal. With some details suppressed, Figure 1 visualizes the execution flow of our main security experiment SEC/I, which acts as an analysis harness for an interface $\mathcal{I}$, game $\mathcal{G}$, and adversary $\mathcal{A}$. The experiment first generates the public and secret keys (pk, sk) as specified by $\mathcal{I}$, then runs $\mathcal{A}$ on input of pk and with access to oracles **Init**, **Call**, and **Final**, which are used by $\mathcal{A}$ to "play" the game $\mathcal{G}$. The game is comprised of three algorithms: the first, $\mathcal{G}$.Init, takes pk as input and outputs the game's initial state; the second, $\mathcal{G}$.Call, specifies the capabilities of $\mathcal{A}$ in the game and advances the state in response to its queries; and the last, $\mathcal{G}$.Final, computes the game's winning condition and outputs a bit *win*. Both $\mathcal{G}$.Call and $\mathcal{G}$.Final are given access to $\mathcal{I}$ for performing secret key operations, and the adversary is given direct access to $\mathcal{I}$ via a fourth oracle **Op**. As usual [12], the adversary must call **Init** first and **Final** last; the outcome of the experiment is the value of *win*.

The central goal of our work is to measure the security "gap" between this and the "usual setting" in which the underlying interface is only used for the target application. This setting is formalized by the SEC experiment, which is defined just like SEC/I, except the adversary is denied access to **Op**. We will formalize both experiments in Section 4.

Context separability. Security in our setting often requires a property we call context separability. Loosely, a context-separable interface is one whose operations can be bound to the context in which they are used. When *context separation* is enforced, this binding prevents context-separable *games* from interacting in unintended ways. Let us consider an illustrative example. TLS is designed to prevent signatures produced in the context of the protocol from being used in other applications, and vice versa. To accomplish this, whenever a message is to be signed, it is signed together with a short *context string* that uniquely identifies the protocol version and the signer (i.e., the client or server, see [54, Section 4.4.3]). This makes it unlikely that another protocol would *inadvertently* produce a signature that could be used in TLS, but nothing about the protocol or the signature scheme ensures this; depending on how signing operations are exposed and whether key separation is enforced, this could lead to practical cross-protocol attacks [15].

As reflected in both our syntax and security notions, our framework sheds formal light on the affect of these design challenges on security. In addition to the secret key and operand, an interface is formalized to take as input a context string ctx, which is meant to uniquely identify the application making the API call; correspondingly, a game is initialized with context that is meant to uniquely identify it. In the SEC/I experiment, the game $\mathcal{G}$ is initialized with an adversarially chosen *game context* string α, which the adversary may not use for its interface queries. (See Figure 1.) This is akin to enforcing non-repeating nonces in the security experiment for symmetric encryption; in practice, it is an operational requirement that the environment must enforce.

On the role of context separation. The high-level goal of our work is to provide a framework for reasoning about the security of interfaces that expose secrets to applications. We uncover context separability as a useful design pattern for achieving security in the presence of key reuse. In fact, this operational requirement can be seen as a generalization of key separation; an interface that enforces key separation by exposing a unique key for each unique application (identified by a context string) it intends to support. But when doing so is infeasible, interfaces and their applications can be designed so that reuse is secure as long as context separation is enforced.

We stress that context separation is not essential to security in the presence of key reuse. We could have formalized other operational requirements; it may suffice to ensure that no single operation is used in multiple applications, or that distinct applications provide distinct inputs, etc. However, our choice to enforce context separation in the SEC/I experiment was not arbitrary. First and foremost, it reflects a design pattern often explicit (but sometimes implicit) in real standards, two of which we analyze in this paper (EdDSA and Noise). Second, it is our hope that clarifying this simple requirement will reduce the complexity inherent to protocol design.

A composition theorem. To measure the "gap" between SEC and SEC/I—that is, to measure the security impact of exposing the underlying interface—in Section 4.2 we formulate and prove sufficiency of a condition under which security in the former sense implies security in the latter. The GAP1 experiment is associated to an interface $\mathcal{I}$, a game $\mathcal{G}$, a simulator $\mathcal{S}$, and a distinguisher $\mathcal{D}$. The experiment allows $\mathcal{D}$ to play the game via **Init**, **Call**, and **Final** as above; likewise, the adversary can query the interface via **Op**. In the "real" world, **Op** exposes $\mathcal{I}$, but in the "simulated" world, the distinguisher's queries are evaluated by $\mathcal{S}$, which is given the public key but *no* access to $\mathcal{I}$. The adversary's goal is to distinguish between these two worlds. We show that for any $\mathcal{I}$ and $\mathcal{G}$, if $\mathcal{I}$ is both SEC and GAP1 secure for $\mathcal{G}$, then $\mathcal{I}$ is also SEC/I secure for $\mathcal{G}$ (Theorem 1(i)). Thus, proving GAP1 security of $\mathcal{I}$ for $\mathcal{G}$ will be our primary goal, as it succinctly characterizes conditions under which it is safe to compose applications that share the same interface.

We also consider the security impact of changing an interface, by, for example, exposing additional operations on the key. The GAP2 experiment is similar to GAP1, except it involves a *pair* of interfaces $(\mathcal{I}^1, \mathcal{I}^0)$. In the "real" world, both the game and distinguisher are given oracle access to $\mathcal{I}^1$; in the "simulated" world, the game is given an oracle for $\mathcal{I}^0$ and the distinguisher's **Op** queries are answered by the simulator, which is also given an oracle for $\mathcal{I}^0$. We prove that if $(\mathcal{I}^1, \mathcal{I}^0)$ is GAP2 secure for $\mathcal{G}$ and $\mathcal{I}^0$ is SEC/I secure for $\mathcal{G}$, then so is $\mathcal{I}^1$ (Theorem 1(ii)). We also formulate a *necessary* condition, wGAP2, that allows us to characterize key operations that are not generally safe to expose in an interface.

Application to discrete log interfaces. We apply our framework to various *discrete log (DL) interfaces*, whose key pairs are $(p = g^s, s)$ where g is the generator of a finite group. They are so named because the security of their applications is predicated on the hardness of computing discrete logarithms (in particular, $s = \log_g p$) in the given group. They are particularly interesting in our setting because they admit a wide variety of primitives and protocols.

Diffie-Hellman and EdDSA. A well-known design challenge for DL interfaces is avoiding accidental exposure of a static Diffie-Hellman (DH) oracle [3, 20]: given p and an oracle that on input of q returns q^s, there is an algorithm [19] for computing s that is much faster than generic DL [53]. As a first exercise of our framework, we rule out the security of (inadvertently) exposing static DH in *any* DL interface by proving wGAP2 insecurity of their composition (Section 5.1). We then consider the security of the EdDSA signature scheme [14] in our setting (Section 5.2). The standardized version of this algorithm [35] admits variants that are context separable, allowing us to prove in the random oracle model (ROM) [11] that the signing operation is GAP1 secure for any game in which all signing and verification operations use the game context. We also show (in the ROM) that exposing the signing operation of *any* EdDSA variant in a DL interface that meets certain requirements is GAP2 secure in general.

Noise. Having addressed the security of these relatively simple operations, in Section 6 we turn to analyzing Noise [52], a framework for designing two-party secure-channel protocols. Participants in these protocols negotiate and execute *handshake patterns*, which define the sequence of messages sent between them and thereby the security of the communication channel they establish. Our results for Noise are largely positive. With a simple tweak of the processing rules, we are able to prove GAP1 security of our interface while making only minimal (and natural) assumptions about the target application. This implies, in particular, that all handshake patterns that can be executed by our interface are jointly secure (up to context separation). We cannot support all patterns, however, because some give rise to GAP1 distinguishing attacks in any interface that could be used to implement them. As a result of these limitations, our analysis leaves the security of key-reuse in Noise *as it is* an open question. Nevertheless, our work shows that Noise's approach to protocol design makes it is possible to reason about protocol interactions in a very general way.

In our analysis of Noise, we model the key-derivation function HKDF [38] as a random oracle (RO). To justify this, and building upon previous results [24], we prove that HKDF is indifferentiable [47] from an RO when the underlying hash function is modeled as an RO. This result appears in Appendix B. Finally, we directly address the composition of EdDSA and Noise in Section 6.4.

Limitations of the framework. Our syntax for games is such that a wide variety of security goals can be expressed with them. However, the execution semantics of games in the SEC/I experiment excludes some important settings, including the multi-user setting [13] and those captured by multi-stage adversaries [55]. In Section 4 we will briefly discuss how to formalize these settings as extensions to the SEC/I experiment. In addition, our interfaces are all *stateless*, which we found necessary for composition in general. (This is in line with prior works that grapple with related problems [55].)

Related work. Our framework generalizes the setting of Shrimpton, Stam, and Warinschi [59], who study HSMs implementing the PKCS#11 standard for cryptographic APIs [31]. Their formulation of a "primitive" is closely related to our formulation of interfaces, and their framework allows for expressing arbitrary security goals for primitives, as ours does for interfaces. One important distinction is that our interfaces are only meant to expose operations on secret keys; public key operations such as encryption or signature verification are specified as part of the game. Our attack model is also much stronger than theirs, since it exposes the interface used in the game to the adversary.

Our security goals are reminiscent of joint security, and many of the proof techniques we use here are borrowed from that area [34, 22]. However, our notions are ultimately incompatible with theirs. To adapt our framework to the consideration of joint security one would partition the set of operations exposed by the interface into those available to the target system (i.e., the game) and those available to the adversary.

The GAP2 notion can be viewed as a restricted form of indifferentiability [47]. In particular, the GAP2 experiment for $(\mathcal{I}_1, \mathcal{I}_0)$, $\mathcal{G}$, adversary $\mathcal{A}$, and simulator $\mathcal{S}$ is equivalent to the indifferentiability of $(\mathcal{I}_1, \mathcal{I}_0)$ with respect to the *specific* distinguisher $\mathcal{D}$ that is the composition of $\mathcal{G}$ and $\mathcal{A}$ prescribed by the GAP2 experiment. To be clear, this does *not* allow us to directly use the indifferentiability composition theorem. Our own result is about composing game $\mathcal{G}$ with interfaces $\mathcal{I}_1$ and, separately, $\mathcal{I}_0$; and although our composition theorem looks quite similar to [55, Theorem 1], the things being composed are not the same.

Examples of API-design flaws. There are several, well-documented examples of API-design flaws leading to vulnerabilities in deployed systems. Degabriele et al. [22] provide an analysis of the EMV standard for credit-card payments. To reduce overhead in this highly constrained environment, the interface permits signing and decryption operations involving the same RSA secret key. Degabriele et al. exhibit a practical forgery attack that uses the decryption operation.

An analysis by Künnemann et al. [41] points out a flaw in the API for Yubico's YubiHSM that admits an oracle for a blockcipher keyed by the same key used to encrypt in CBC mode, leading to a simple plaintext-recovery attack.

In recent years, Intel and other chip manufacturers have moved to develop, implement, and deploy protocol standards allowing for remote attestation of the state of a host, often called trustworthy computing. Since the host is often an end-user system (e.g., a laptop or cellphone), performing an attestation in a privacy-preserving manner is paramount. The TPM ("trusted platform module") standard exposes an interface for direct anonymous attestation (DAA) [18] in a variety of protocols. The is made possible by a very flexible API that, unfortunately, leads to security issues [3]. Designing an interface for TPM that is both sufficiently flexible and secure has proved challenging. A solution was recently devised by Camenish et al. [20]. Their contribution is two-fold: first, they modify the protocol so that it can be proven secure (analyses of prior versions of TPM remote attestation were erroneous); and second, they redesign the interface to mitigate the static DH oracle.

Keyless SSL is a protocol deployed by Cloudflare used to proxy TLS connections between clients and servers without the need to have the server's signing key on premise. In this protocol, the server exposes a signing API to mutually authenticated peers. To sign a handshake with a client, the server sends the message to be signed to the server, who responds with the signature. Instead of operating on the message M itself, the API operates on a *hash* of the message $H(M)$, which is an intermediate value in certain signature schemes. Bhargavan et al. [15] show that this interface admits a cross-protocol attack with QUIC, another widely-deployed transport-security protocol. In the absence of mutual authentication of the peer (which Keyless SSL provides), their attack would allow anyone to impersonate a QUIC server.

API usability. The current work could be viewed as a formal treatment of a small piece of the much larger problem of API usability. Since Adams and Sasse's seminal work "Users are not the Enemy" [5], usability being necessary for the deployment of secure systems has emerged as an axiom in the research community. Recently, this focus on usability has shifted from the end user to the developer of the system [33]: specifically, towards understanding how developers think about and use APIs, how API misuse leads to vulnerabilities, and how to design APIs that are easy to use securely and hard to use insecurely. Much of this work has been catalyzed by vulnerabilities that are generally believed to result from the complexity and poor quality of existing cryptographic APIs [30, 27, 26]. User studies have been especially fruitful in corroborating this belief [4, 51]. In their study of the psychological factors involved in the development of vulnerable code, Oliviera et al. [50] point out that developers tend to trust that APIs are secure, which leads to blind spots in their mental model of how the API is to be used securely.

2 Pseudocode and Conventions

This section enumerates our conventions for pseudocode, algorithms, adversaries, and experiments. The reader may wish to skip this section and refer to it later as needed.

Pseudocode. Our pseudocode is based on Rogaway and Stegers [56]. Variables are statically typed. Available types are **set** (a set), **tup** (a tuple), **bool** (an element of $\{0,1\}$), **int** (an element of $\mathbb{Z}$), and **str** (an element of $\{0,1\}^*$). In general, if $X \in \mathcal{X}$, then we say that X has type $\mathbf{elem}_{\mathcal{X}}$. Variables are declared with the keyword **dec**, e.g., **dec** **int** x; **str** A. Variables need not be explicitly declared, in which case their type must be inferable from their initialization (i.e., the first use of the variable in an assignment statement). There are two *compound* types. The first is associative arrays, denoted by "[ ]", which map tuples (that is, a finite sequence of quantities of any type) to values of a specific type. For example, **dec** **str** $\pi[\,]$ declares an associative array π whose values are strings. We let $\pi[k]$ and π_k denote the value in π associated with k. The second is **struct**, which is used to recursively define new types; see Figure 7 for an example. We will also refer to the type of a procedure (i.e., an algorithm) by its interface. For instance, the type $\mathcal{A}(\mathbf{str}\, X, Y) \mapsto (\mathbf{int}\, i, \mathbf{str}\, A)$ indicates that $\mathcal{A}$ takes as input a pair of strings (X, Y) and outputs an integer i and a string A.

Nil and bottom. Uninitialized variables implicitly have the value $\diamond$, read "nil". If a variable of one type is set to a value of another type, then the variable takes the value $\diamond$. The symbol $\diamond$ is interpreted as $\emptyset$ in an expression involving sets, as the 0-length tuple in an expression involving tuples, as 0 (i.e., false) in a boolean expression, as 0 in an expression involving integers, and as ε in an expression involving strings. A non-**bool** variable X is interpreted as "$(X \neq \diamond)$" (i.e., "X is defined") in a boolean expression. If X is an associative array, then $X \leftarrow \diamond$ "resets" the array so that $X_k = \diamond$ for all k. Likewise, if $\mathcal{X}$ is a **struct**, then $X \leftarrow \diamond$ sets each field to $\diamond$. The symbol $\perp$, read "bottom", can be assigned to any variable regardless of type. Unlike $\diamond$, its interpretation in an expression is always undefined, except that $X = \perp$ and $\perp = X$ should evaluate to true just in case the previous assignment to X was $\perp$. (We remark that $\perp$ has the usual semantics in cryptographic pseudocode.)

Represented groups. We say that a group $\mathbb{G}$ is *represented* if $\diamond \notin \mathbb{G}$. We define an additional type, $\mathbf{elem}_{\mathbb{G}}$, parameterized by a represented group $\mathbb{G}$. We emphasize that, unlike **set**, **tup**, **bool**, **int**, or **str**, using the symbol $\diamond$ in an expression involving values of this type is not well-defined, since $\diamond$ has no interpretation as an element of $\mathbb{G}$.

Refined types. Variable declarations may be written as set-membership assertions. For example, **dec** **int** s; $\mathbf{elem}_{\mathbb{G}}\, P$ may be written like **dec** $s \in \mathbb{Z}$; $P \in \mathbb{G}$. Where appropriate, these types may also be refined, e.g. **dec** $s \in \mathbb{N}$, as long as the set specified by the refined type contains $\diamond$.

String and tuple operations. Let $|X|$ denote the length of a string (or tuple) X. We denote the i-th element of X by X_i or $X[i]$. We define $X \,\|\, Y$ to be the concatenation of X with string (or tuple) Y. Let $X[i{:}j]$ denote the sub-string (or sub-tuple) $X_i \,\|\, \cdots \,\|\, X_j$ of X. If $i \notin [1..j]$ or $j \notin [i..|X|]$, then define $X[i{:}j] = \diamond$. Let $X[i{:}] = X[i{:}|X|]$ and $X[{:}j] = X[1{:}j]$.

Encoding of types. A value of any type can be encoded as a string. We will not define it explicitly, but assume it possesses the following properties. Let $\underline{x_1, \ldots, x_m}$ denote the encoding of a tuple $(x_1, \ldots, x_m)$ as a string. Decoding is written as $\underline{x_1, \ldots, x_m} \leftarrow X$ and works like this (slightly deviating from [56, Section 2]): if there exist $y_1, \ldots y_n$ such that $X = \underline{y_1, \ldots, y_n}$, $m = n$, and each y_i has the same type as x_i, then set $x_i \leftarrow y_i$ for each $1 \leq i \leq m$. Otherwise, set $x_i \leftarrow \diamond$ for each $1 \leq i \leq m$. Finally, let $\underline{x}_n$ denote the encoding of non-negative integer $x \geq 0$ as an n-bit string. We write $\underline{x}_n \leftarrow X$ to denote decoding X as an n-bit, non-negative integer and assigning it to x.

Passing variables by reference. It is customary in cryptographic pseudocode to pass all variables by *value*; we also permit variables to be passed by *reference*. (This idea is due to Rogaway and Stegers [56], but our semantics deviates from theirs.) Specifically, variables passed to procedures may be embellished with the symbol "&". If the variable appears on the left hand side of an assignment statement, then this immediately changes the value of the variable; when used in an expression, the variable is treated as its value. A procedure's interface makes explicit each input that is passed by reference. For example, in a procedure $\mathcal{A}(\&\mathbf{int}\, x, \mathbf{int}\, y) \mapsto \mathbf{int}\, z$, variable y is passed by value, while x is passed by reference. After executing $x, y \leftarrow 0$; $z \twoheadleftarrow \mathcal{A}(\&x, y)$, the value of x may be non-0, but y is necessarily equal to 0.

Algorithms, experiments, and adversaries. Algorithms are randomized unless stated otherwise. An algorithm is t-time if for every choice of random coins, the algorithm halts in at most t time steps.[1] When an algorithm $\mathcal{A}$ is deterministic we write $y \leftarrow \mathcal{A}(x)$ to denote executing $\mathcal{A}$ on input of x and assigning its output to y; if $\mathcal{A}$ is randomized, then we write $y \twoheadleftarrow \mathcal{A}(x)$. Let $[\mathcal{A}(x)]$ denote the set of possible outputs of $\mathcal{A}$ when run on input x. Algorithms may have access to one or more oracles, written as superscripts, e.g., $y \twoheadleftarrow \mathcal{A}^{\mathcal{O},\cdots}(x)$. When this notation becomes cumbersome we may write $y \twoheadleftarrow \langle \mathcal{A} \colon \mathcal{O}, \ldots \rangle(x)$ instead. When we specify a procedure in pseudocode, if the procedure halts without an explicit **ret**-statement (i.e., a "return" statement), then it returns $\perp$.

We regard security experiments as algorithms whose output is always a bit. If "XXX" is an experiment associated with an adversary $\mathcal{A}$, we write $\mathbf{Exp}^{\mathrm{xxx}}(\mathcal{A})$ to denote the event that the experiment is run with $\mathcal{A}$ and the output is 1, i.e., $\Pr\big[\,\mathbf{Exp}^{\mathrm{xxx}}(\mathcal{A})\,\big]$ denotes the probability that XXX run with $\mathcal{A}$ outputs 1, where the probability is over the coins of XXX and $\mathcal{A}$. An adversary is an algorithm associated to a security experiment in which it is executed exactly once. (Thus, in this paper we restrict ourselves to the single-stage adversary setting [55].) Our convention will be that an adversary is t-time if its experiment is t-time. That is, an XXX-adversary $\mathcal{A}$ is t-time if $\mathbf{Exp}^{\mathrm{xxx}}(\mathcal{A})$ is t-time.

Miscellaneous. Logarithms are base-2 unless the base is given explicitly. If $\mathcal{X}$ is a set, then we write $x \twoheadleftarrow \mathcal{X}$ to denote sampling x randomly from $\mathcal{X}$ according to some distribution associated to $\mathcal{X}$; if $\mathcal{X}$ is finite and the distribution is unspecified, then it is uniform. For every integer $n \geq 1$ let $[n]$ denote the set $\{1, \ldots, n\}$. Let $\operatorname{Dom} f$ denote the domain of function f and let $\operatorname{Rng} f$ denote its range. Let $\Delta(U, V)$ denote the statistical distance between random variables U and V. We write $X \preceq Y$ if string (or tuple) X is a prefix of string (or tuple) Y (i.e., $(\exists\, T)\, X \,\|\, T = Y$). We write $S \sim X$ if $|S| > 0$ and S is a sub-string (or sub-tuple) of X (i.e., $|S| \leq |X|$ and $(\exists\, i)\, X[i{:}i + |S|] = S$).

Pseudocode in this paper implicitly treats strings over the alphabet $\Sigma = \{\mathsf{A}, \ldots, \mathsf{Z}, \mathsf{a}, \ldots, \mathsf{z}, \mathsf{0}, \ldots \mathsf{9}, \_\}$ as bit strings (i.e., of type **str**). We do so by fixing an injection $f : \Sigma^* \to \{0, 1\}^*$ and write s instead of $f(s)$. For example, when we write "dh" or "sig", we really mean "$f(\mathsf{dh})$" or "$f(\mathsf{sig})$". Such an f is easy to construct; it might map each element of Σ to its ASCII encoding.

3 Interfaces and Games

In this section we define the syntax for *interfaces* and *games*, which are the fundamental components of our formalization. A game captures an attack model (the capabilities and goals of an adversary) as well as an

---

[1] What constitutes a "time step" depends on the model of computation, which we leave implicit.

```
$\boxed{\mathbf{Exp}^{\text{sec/i}}_{\mathcal{I},\mathcal{G}}(\mathcal{A})}$ / $\mathbf{Exp}^{\text{sec}}_{\mathcal{I},\mathcal{G}}(\mathcal{A})$
    dec str $sk, st, \alpha$; bool $win$
    $(pk, sk) \twoheadleftarrow \mathcal{I}.\text{Gen}(\,)$
    $\boxed{\langle \mathcal{A}\colon \textbf{Init}, \textbf{Final}, \textbf{Call}, \textbf{Op}\rangle(pk)}$
    $\langle \mathcal{A}\colon \textbf{Init}, \textbf{Final}, \textbf{Call}\rangle(pk)$
    ret $win$

Init$(ctx)$
    $(st, out) \twoheadleftarrow \mathcal{G}.\text{Init}(pk, ctx)$
    $\alpha \leftarrow ctx$; ret $out$

Final$(in)$
    $win \twoheadleftarrow \mathcal{G}.\text{Final}^{\mathcal{I}.\text{Op}(sk,\cdot,\cdot,\cdot)}(st, in)$
    ret $win$

Call$(in)$
    ret $\mathcal{G}.\text{Call}^{\mathcal{I}.\text{Op}(sk,\cdot,\cdot,\cdot)}(\&st, in)$

Op$(ctx, op, in)$
    if $ctx = \alpha$ then ret $\perp$
    ret $\mathcal{I}.\text{Op}(sk, ctx, op, in)$
```

Figure 2: The SEC/I and SEC experiments for interface $\mathcal{I}$, game $\mathcal{G}$, and adversary $\mathcal{A}$.

---

intended use of cryptographic operations that are provided (via black-box calls) by an interface. Typically, this use will be to realize some cryptographic scheme (i.e., primitive or protocol) that is under attack.

Definition 1 (Interfaces)**.** An *interface* is a pair of algorithms $\mathcal{I} = (\text{Gen}, \text{Op})$ defined as follows:

- $\text{Gen}(\,) \mapsto \textbf{str}\ pk, sk$. The *key generator* outputs pair of key strings.
- $\text{Op}(\textbf{str}\ sk, ctx, op, in) \mapsto \textbf{str}\ out$. The *key operator* exposes operations involving the key sk. It takes as input the context ctx, the operation identifier op, and the operand in, and it outputs the result out.

For compactness, we may denote $\mathcal{I}.\text{Op}(sk, ctx, op, in)$ by $\mathcal{I}_{sk}(ctx, op, in)$ in the remainder. ⧫

In our security experiments, the "public key" pk will be made available to all parties, but the "secret key" sk will be kept private by the interface. We note that $pk = \varepsilon$ is allowed, so that symmetric-key operations are within scope.

Definition 2 (Games)**.** A *game* is a triple of algorithms $\mathcal{G} = (\text{Init}, \text{Call}, \text{Final})$ defined as follows:

- $\text{Init}(\textbf{str}\ pk, \alpha) \mapsto \textbf{str}\ st, out$. This is the game *initiator*. It takes as input the public key pk and game context α and outputs the initial state st and a string out.
- $\text{Call}^{\mathcal{O}}(\&\textbf{str}\ st, \textbf{str}\ in) \mapsto \textbf{str}\ out$. The *caller* is used to advance the state of an already initialized game. It abstracts all oracle queries except initialization and finalization. The first input is a reference to the game state, which may be updated as a side-effect of invoking the caller; the interpretation of second input is up to the game. The caller expects access to an oracle $\mathcal{O}$, which we will call the *interface oracle*. It takes as input three strings and returns one.
- $\text{Final}^{\mathcal{O}}(\textbf{str}\ st, in) \mapsto \textbf{bool}\ r$. The *finalizer* is used to decide if a game is in a winning state. Its inputs are the game state st and a string in, which is used to compute the winning condition. Oracle $\mathcal{O}$ is as defined for the caller.

For compactness, we occasionally denote $\mathcal{G}.\text{Call}^{\mathcal{O}}(\&st, in)$ by $\mathcal{G}^{\mathcal{O}}_{st}(in)$. We say that $\mathcal{G}$ is *c-bound* if the caller and finalizer each make at most c calls to $\mathcal{O}$ during any execution of the algorithm. ⧫

4 Security Under Exposed Interface Attack

The goal of this work is to understand the security of cryptographic schemes when they are realized by an interface that may also be exposed to other, possibly insecure or (or even malicious) applications. The following experiment (SEC/I) captures this formally, allowing us prove or disprove security of a scheme (both codified by a game $\mathcal{G}$) when a given interface $\mathcal{I}$ is callable by both the game $\mathcal{G}$ and the adversary $\mathcal{A}$. An adversary in this experiment is said to be mounting an *exposed interface attack* on $\mathcal{G}$. We also define an experiment that captures the usual setting in which the adversary does not have direct access to the interface (SEC).

Definition 3 (SEC/I and SEC security)**.** Figure 2 defines two security experiments: SEC/I includes the boxed statement (but not the shaded statement), and SEC includes the shaded statement (but not the boxed one). Both experiments begin by running the key generator $\mathcal{I}$.Gen and executing the adversary $\mathcal{A}$ on input of the public key and with access to four oracle procedures:

- **Init** initializes $\mathcal{G}$ by calling the initiator $\mathcal{G}$.Init on the public key and the game context chosen by $\mathcal{A}$ and returns the output *out* of the initiator.
- **Call** advances the game by invoking the caller $\mathcal{G}$.Call on input *in* provided by $\mathcal{A}$ and with oracle access to the interface $\mathcal{I}.\mathrm{Op}(sk, \cdot, \cdot, \cdot)$. It returns the output *out* of the caller.
- **Op** exposes $\mathcal{I}.\mathrm{Op}(sk, \cdot, \cdot, \cdot)$ to $\mathcal{A}$ directly with the restriction that each query use a context string *ctx* that is different from the game context used to initialize the game.
- **Final** finalizes $\mathcal{G}$ by running the finalizer $\mathcal{G}$.Final on input *in* provided by $\mathcal{A}$ and setting *win* to the output and returning the value of *win* to $\mathcal{A}$.

The outcome of the experiment is the value of *win* when $\mathcal{A}$ halts. A valid SEC/I adversary makes a single query to **Init**, this being its first; it may then make any number of queries to **Call** and **Op**.[2] It completes its execution by making a single query to **Final**. We define the advantage of a (valid) SEC/I-adversary $\mathcal{A}$ in attacking $\mathcal{I}$ with respect to $\mathcal{G}$ as

$$\mathbf{Adv}^{\text{sec/i}}_{\mathcal{I},\mathcal{G}}(\mathcal{A}) = \Pr\left[\, \mathbf{Exp}^{\text{sec/i}}_{\mathcal{I},\mathcal{G}}(\mathcal{A}) \,\right].$$

We call a SEC/I adversary (t, q_G, q_I)-resource if it is t-time and makes at most q_G and q_I queries to **Call** and **Op** respectively. We define the maximum advantage of any $\boldsymbol{r}$-resource SEC/I-adversary as $\mathbf{Adv}^{\text{sec/i}}_{\mathcal{I},\mathcal{G}}(\boldsymbol{r})$. SEC security of $\mathcal{I}$ with respect to $\mathcal{G}$ is defined in kind, except that **Op** is not given to $\mathcal{A}$. We denote the advantage of SEC-adverseary $\mathcal{A}$ in attacking $\mathcal{I}$ with respect to $\mathcal{G}$ by $\mathbf{Adv}^{\text{sec}}_{\mathcal{I},\mathcal{G}}(\mathcal{A}) = \Pr\left[\, \mathbf{Exp}^{\text{sec}}_{\mathcal{I},\mathcal{G}}(\mathcal{A}) \,\right]$, and we define $\mathbf{Adv}^{\text{sec}}_{\mathcal{I},\mathcal{G}}(\boldsymbol{r})$ as above. Informally, we say that $\mathcal{I}$ is SEC/I (resp. SEC) secure for $\mathcal{G}$ if every efficient SEC/I (resp. SEC) adversary has small advantage.

Finally, if each of $\mathcal{G}$.Call's and $\mathcal{G}$.Final's interface queries is a triple (α, op, in) such that α is the context with which the game was initialized, then we say $\mathcal{G}$ is *regular* for SEC/I (resp. SEC). ⧫

Regular games and context separation. We remark that a game being regular is a property of the execution semantics of the game in the experiment, and not a syntactic property of the game itself. This is because an experiment might execute the game differently; for example, instead of invoking the initiator before the caller, the experiment could invoke the caller with state $st = \varepsilon$ each time. This may sound silly, but we have not given a syntactic condition on games that excludes this execution semantics. Because all experiments will run the game in the same way, we silently extend this definition of regularity to all experiments in the remainder of the paper. In our analyses in Sections 5 and 6, we will prove SEC/I security with respect to regular games. This condition is sufficient for enforcing context separation between operations performed by the adversary via direct access to the interface and those performed by the game.

Indistinguishability variants. We note that our definitions of SEC/I and SEC advantage are not appropriate for every game. For example, $\mathcal{G}$ might be a bit-*guessing* game (e.g., IND-CCA) in which the initiator flips a coin and the finalizer interprets its input as the adversary's guess. In order to normalize the adversary's advantage in such games, we define the IND-SEC/I advantage of SEC/I-adversary $\mathcal{A}$ as $\mathbf{Adv}^{\text{ind-sec/i}}_{\mathcal{I},\mathcal{G}}(\mathcal{A}) = 2\mathbf{Adv}^{\text{sec/i}}_{\mathcal{I},\mathcal{G}}(\mathcal{A}) - 1$. (Similarly for IND-SEC.)

Limitations of the SEC/I experiment. The execution semantics of our experiments restrict our setting to single-staged games as defined by Ristenpart, Shacham and Shrimpton [55]. In particular, the adversary may keep arbitrary state throughout its run. This means that we cannot address, for example, the SEC/I security of certain games capturing properties of deterministic or hedged public-key encryption [8, 9], key-dependent message security [16], or security against related-key attacks [10] when one allows related-key functions that can depend on some ideal primitive [6]. Modeling multi-stage adversaries would require defining a new experiment in which the execution semantics of each stage is precisely defined. That said, many important

---

[2] Disallowing **Op** queries prior to **Init** is necessary for enforcing context separation. This restriction could be lifted by, say, allowing pre-**Init** access to **Op**, but demanding that none of these queries uses the (adversarially chosen) game context α.

```
$\mathbf{Exp}^{\mathrm{gap1}}_{\mathcal{I},\mathcal{G}}(\mathcal{S},\mathcal{D})$
  $\mathbf{dec\ str}\ sk, st, \sigma, \alpha;\ b \twoheadleftarrow \{0,1\}$
  $(pk, sk) \twoheadleftarrow \mathcal{I}.\mathrm{Gen}(\,);\ \sigma \twoheadleftarrow \mathcal{S}.\mathrm{Init}(pk)$
  $d \twoheadleftarrow \langle \mathcal{D}\colon \mathbf{Init}, \mathbf{Final}, \mathbf{Call}, \mathbf{Op} \rangle(pk)$
  $\mathbf{ret}\ (d = b)$

$\mathbf{Init}(ctx)$
  $(st, out) \twoheadleftarrow \mathcal{G}.\mathrm{Init}(pk, ctx)$
  $\alpha \leftarrow ctx;\ \mathbf{ret}\ out$

$\mathbf{Final}(in)$
  $\mathbf{ret}\ \mathcal{G}.\mathrm{Final}^{\mathcal{I}_{sk}}(st, in)$

$\mathbf{Call}(in)$
  $\mathbf{ret}\ \mathcal{G}.\mathrm{Call}^{\mathcal{I}_{sk}}(\&st, in)$

$\mathbf{Op}(ctx, op, in)$
  $\mathbf{if}\ ctx = \alpha\ \mathbf{then\ ret}\ \perp$
  $\mathbf{if}\ b = 1\ \mathbf{then\ ret}\ \mathcal{I}_{sk}(ctx, op, in)$
  $\mathbf{ret}\ \mathcal{S}.\mathrm{Op}^{\perp}(\&\sigma, ctx, op, in)$
```

```
$\mathbf{Exp}^{\mathrm{gap2}}_{\mathcal{I}^1,\mathcal{I}^0,\mathcal{G}}(\mathcal{S},\mathcal{D})$
  $\mathbf{dec\ str}\ sk, st, \sigma, \alpha;\ b \twoheadleftarrow \{0,1\}$
  $(pk, sk) \twoheadleftarrow \mathcal{I}^b.\mathrm{Gen}(\,);\ \sigma \twoheadleftarrow \mathcal{S}.\mathrm{Init}(pk)$
  $d \twoheadleftarrow \langle \mathcal{D}\colon \mathbf{Init}, \mathbf{Final}, \mathbf{Call}, \mathbf{Op} \rangle(pk)$
  $\mathbf{ret}\ (d = b)$

$\mathbf{Init}(ctx)$
  $(st, out) \twoheadleftarrow \mathcal{G}.\mathrm{Init}(pk, ctx)$
  $\alpha \leftarrow ctx;\ \mathbf{ret}\ out$

$\mathbf{Final}(in)$
  $\mathbf{ret}\ \mathcal{G}.\mathrm{Final}^{\mathcal{I}^b_{sk}}(st, in)$

$\mathbf{Call}(in)$
  $\mathbf{ret}\ \mathcal{G}.\mathrm{Call}^{\mathcal{I}^b_{sk}}(\&st, in)$

$\mathbf{Op}(ctx, op, in)$
  $\mathbf{if}\ ctx = \alpha\ \mathbf{then\ ret}\ \perp$
  $\mathbf{if}\ b = 1\ \mathbf{then\ ret}\ \mathcal{I}^1_{sk}(ctx, op, in)$
  $\mathbf{ret}\ \mathcal{S}.\mathrm{Op}^{\mathcal{I}^0_{sk}}(\&\sigma, ctx, op, in)$
```

```
$\mathbf{Exp}^{\mathrm{wgap2}}_{\mathcal{I}^1,\mathcal{I}^0,\mathcal{G}}(\mathcal{S},\mathcal{D})$
  $\mathbf{dec\ str}\ sk, st, \sigma;\ b \twoheadleftarrow \{0,1\}$
  $(pk, sk) \twoheadleftarrow \mathcal{I}^b.\mathrm{Gen}(\,);\ \sigma \twoheadleftarrow \mathcal{S}.\mathrm{Init}(pk)$
  $d \twoheadleftarrow \mathcal{D}^{\mathbf{Op}}(pk);\ \mathbf{ret}\ (d = b)$

$\mathbf{Op}(ctx, op, in)$
  $\mathbf{if}\ b = 1\ \mathbf{then\ ret}\ \mathcal{I}^1_{sk}(ctx, op, in)$
  $\mathbf{ret}\ \mathcal{S}.\mathrm{Op}^{\mathcal{I}^0_{sk}}(\&\sigma, ctx, op, in)$
```

Figure 3: Top-left: The GAP1 experiment for interface $\mathcal{I}$, game $\mathcal{G}$, simulator $\mathcal{S}$, and adversary $\mathcal{D}$. Top-right: The GAP2 experiment for interfaces $\mathcal{I}^1$ and $\mathcal{I}^0$, $\mathcal{G}$, $\mathcal{S}$, and $\mathcal{D}$. Bottom: The wGAP2 experiment for $(\mathcal{I}^1, \mathcal{I}^0)$ respectively.

security goals are captured by single-stage games, including classic notions of privacy (IND-CCA) of encryption schemes, unforgeability (UF-CMA) of signatures, entity authentication (EA), and authenticated key exchange (AKE).

Our experiments allow for modeling corruption of secret state associated with the game (for example, when modeling forward secrecy of an AKE protocol), but not of the secret key exposed by the interface. (Unless the interface provides an operation that returns *sk*, in which case SEC/I security is a lost cause.) Corruption of such long-term secrets held by interfaces could be modeled by introducing a new oracle to the experiment.

Finally, we note that the SEC/I experiment is not well-suited for analyzing games in the multi-user setting. (See related work in Bellare and Tackmann [13] for a nice overview of this area.) One may wish to define an experiment that is played with multiple interfaces, perhaps corresponding to multiple parties. One way to do this would be to "wrap" the adversary's **Call** queries so that it specifies which interface the game should call. However, we will leave the details to future work.

4.1 Simulatability of an Interface

Intuitively, the "gap" between the SEC/I and SEC security of an interface $\mathcal{I}$ with respect to game $\mathcal{G}$ is driven by any extra leverage the attacker gains by interacting with $\mathcal{I}$ directly. In this section, we formalize an experiment that aims to measure the size of this gap for a given $\mathcal{I}$ and $\mathcal{G}$. We also define a related experiment that measures the relative security "gap" between a pair of interfaces $(\mathcal{I}_1, \mathcal{I}_0)$ with respect to a given game. This is particularly useful when the operations permitted by $\mathcal{I}_1$ are a superset of those permitted by $\mathcal{I}_0$. For example, in Section 5, we will use this notion to analyze the change in security when operations are added to an existing interface. Both of these experiments will make use of simulators, so let us first define these.

Definition 4 (Simulators)**.** A simulator $\mathcal{S}$ is a tuple of algorithms $(\mathrm{Init}, \mathrm{Op})$ defined as follows:

- $\mathrm{Init}(\mathbf{str}\, pk) \mapsto \mathbf{str}\, \sigma$. The *initiator* takes as input a public key and outputs the simulator's initial state σ.
- $\mathrm{Op}^{\mathcal{O}}(\&\mathbf{str}\, \sigma, \mathbf{str}\, ctx, op, in) \mapsto \mathbf{str}\, out$. The *operator* takes as input a reference to the simulator state (which it may update as a side-effect) and a triple of strings (ctx, op, in) and outputs a string *out*. Oracle $\mathcal{O}$ is an interface oracle defined just as for games.

In the remainder, we may denote $\mathcal{S}.\mathrm{Op}^{\mathcal{O}}(\&\sigma, ctx, op, in)$ by $\mathcal{S}^{\mathcal{O}}_{\sigma}(ctx, op, in)$. We say that $\mathcal{S}$ is (t, q_I)-resource if each algorithm is t-time and the caller makes at most q_I queries to its oracle. ♦

Definition 5 (GAP1/2 security)**.** Figure 3 defines two experiments: GAP1 and GAP2. Each involves a simulator $\mathcal{S}$, an adversary $\mathcal{D}$, and a game $\mathcal{G}$; GAP1 involves a single interface $\mathcal{I}$, while GAP2 involves a pair interfaces $(\mathcal{I}_1, \mathcal{I}_0)$. Both begin by choosing a challenge bit b at random, executing the key generator ($\mathcal{I}$.Gen in GAP1 and $\mathcal{I}^b$.Gen in GAP2), and initializing the simulator via $\mathcal{S}$.Init on input of the public key. The adversary is then executed on input of the public key and with four oracles:

- **Init**, **Final**, and **Call** execute the game just like in the SEC/I experiment; interface queries are answered by $\mathcal{I}$.Op in GAP1 and $\mathcal{I}^b$.Op in GAP2.
- **Op** processes (ctx, op, in) as follows. If ctx is equal to the game context, then it returns $\perp$ (just as in SEC/I). If $b = 1$, then it returns $\mathcal{I}.\mathrm{Op}(sk, ctx, op, in)$ in GAP1 and $\mathcal{I}^1.\mathrm{Op}(sk, ctx, op, in)$; otherwise it returns $\mathcal{S}.\mathrm{Op}^{\perp}(\&\sigma, ctx, op, in)$ in GAP1 and $\mathcal{S}.\mathrm{Op}^{\mathcal{I}^0_{sk}}(\&\sigma, ctx, op, in)$ in GAP2.[3]

The outcome of the experiment is the bit d output by $\mathcal{D}$ when it halts. A valid GAP1 (resp. GAP2) adversary makes a single query to **Init**, this being its first query; it may then make any number of queries to **Call** and **Op**. It completes its execution by making a single query to **Final**. We define the advantage of a (valid) GAP1-adversary $\mathcal{D}$ in attacking $\mathcal{I}$ with respect to $\mathcal{G}$ as

$$\mathbf{Adv}^{\mathrm{gap1}}_{\mathcal{I},\mathcal{G}}(\mathcal{S}, \mathcal{D}) = 2 \Pr\big[\, \mathbf{Exp}^{\mathrm{gap1}}_{\mathcal{I},\mathcal{G}}(\mathcal{S}, \mathcal{D}) \,\big] - 1 \,.$$

We call an GAP1 adversary (t, q_G, q_I)-resource if it is t-time and makes at most q_G and q_I queries to **Call** and **Op** respectively. We define the maximum advantage of any $\boldsymbol{r}$-resource GAP1 adversary (for a given $\mathcal{I}, \mathcal{G}, \mathcal{S}$) as $\mathbf{Adv}^{\mathrm{gap1}}_{\mathcal{I},\mathcal{G}}(\mathcal{S}, \boldsymbol{r})$. Define $\mathbf{Adv}^{\mathrm{gap2}}_{\mathcal{I}^1,\mathcal{I}^0,\mathcal{G}}(\mathcal{S}, \mathcal{D})$ and $\mathbf{Adv}^{\mathrm{gap2}}_{\mathcal{I}^1,\mathcal{I}^0,\mathcal{G}}(\mathcal{S}, \boldsymbol{r})$ in kind. Informally, we say that $\mathcal{I}$ (resp. $(\mathcal{I}^1, \mathcal{I}^0)$) is GAP1 (resp. GAP2) secure for $\mathcal{G}$ if for every efficient GAP1 (resp. GAP2) adversary $\mathcal{D}$ there exists an efficient $\mathcal{S}$ such that $\mathcal{D}$ has small advantage.

Finally, we say that a simulator is *regular* for GAP1 (resp. GAP2) if each time it is called with input context ctx, each of its interface queries have the form (ctx, op, in) for some $op, in \in \{0,1\}^*$. ♦

4.2 The Composition Theorem

An interface $\mathcal{I}$ being GAP1 secure for $\mathcal{G}$ means that whatever information an SEC/I adversary learns in its attack against $\mathcal{G}$ it can (efficiently) compute on its own without interacting with the **Op** oracle. Thus, if $\mathcal{I}$ is both SEC and GAP1 secure for $\mathcal{G}$, then it should be that $\mathcal{I}$ is also SEC/I secure for $\mathcal{G}$. Relatedly, for any pair of interfaces $(\mathcal{I}^1, \mathcal{I}^0)$ and game $\mathcal{G}$, if $(\mathcal{I}^1, \mathcal{I}^0)$ is GAP2 secure for $\mathcal{G}$ and $\mathcal{I}^0$ is SEC/I secure for $\mathcal{G}$, then $\mathcal{I}^1$ is SEC/I-secure for $\mathcal{G}$, too. Theorem 1 makes these claims precise. To support upcoming results in Sections 5 and 6, we state and prove our composition theorem in the ROM. So, let us first formalize the ROM in our setting.

The ROM. When modeling a function $H : \mathcal{X} \to \mathcal{Y}$ as a random oracle (RO) in an experiment, we declare an associative array $\mathbf{elem}_{\mathcal{Y}}\, \pi[\,]$ and a set $\mathcal{Q}$ (initially empty) and define three oracles: **P**, **Q**, and **R**. The last of these is the usual RO: on input of $X \in \mathcal{X}$, oracle **R** checks to see if π_X is defined (i.e., $\pi_X \neq \diamond$); if not, then it samples π_X from $\mathcal{Y}$ according to the distribution induced on $\mathcal{Y}$ by H. (Usually $\mathcal{Y}$ will be finite and the distribution will be uniform.) Finally, it returns π_X. We call an algorithm *q_R-ro-bound* if it makes at most q_R queries to **R** during any execution; a game, interface, or simulator is q_R-ro-bound if each of its constituent algorithms is q_R-ro-bound. Experiments are lifted to the ROM by providing each named algorithm oracle access to **R**. In addition, each query X to **R** made by the adversary is added to the set $\mathcal{Q}$.

---

[3] The "$\perp$" oracle given to $\mathcal{S}$ denotes the interface oracle that just returns $\perp$ on any query.

Just as we measure an adversary's runtime using the experiment in which it is executed, our convention will be that an adversary's RO-query budget accounts for all queries to $\mathbf{R}$ made by it or any other algorithm (including the simulator) during the course of the experiment. That is, XXX-adversary $\mathcal{A}$ is q_R-ro-bound if $\mathbf{Exp}^{\mathrm{xxx}}(\mathcal{A})$ is q_R-ro-bound. We say an algorithm is $(\boldsymbol{r} \,\|\, q_R)$-resource if it is $\boldsymbol{r}$-resource and q_R-ro-bound. (Note that $\boldsymbol{r} \,\|\, q_R$ is a tuple, since $\boldsymbol{r}$ is a tuple and q_R is a singleton.) Let $\psi : \{0,1\}^* \times \mathcal{X} \to \{0,1\}$ be a function. We say that a game $\mathcal{G}$ is *ψ-ro-regular* (for the associated experiment) if each of its RO queries $X \in \mathcal{X}$ satisfies $\psi(\alpha, X)$, where α is the game context used to initialize it in the experiment. Similarly, we say that an interface $\mathcal{I}$ is ψ-ro-regular if each of $\mathcal{I}$.Op's RO queries $X \in \mathcal{X}$ satisfies $\psi(ctx, X)$, where ctx is the provided context string.

The other two oracles ($\mathbf{P}$ and $\mathbf{Q}$) are used to specify additional powers made available to simulators in security proofs. Oracle $\mathbf{P}$ takes as input a pair $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ and sets $\pi[X] \leftarrow Y$, allowing the simulator to "program" the RO. Oracle $\mathbf{Q}$ simply returns the set $\mathcal{Q}$ of RO queries made by the adversary so far, allowing the simulator to "observe" the adversary's RO queries as it makes them.

Joint security in the ROM. We emphasize that $\mathbf{P}$ and $\mathbf{Q}$ formalize powers of the simulator that are usually left implicit, but are essential to certain proof techniques [34, 22]. The usual approach to proving joint security of cryptosystems is to show that the oracle cryptosystem is efficiently simulatable without knowledge of the secret key. Haber and Pinkas [34] elicit two techniques for doing so in the ROM: the first requires the ability to observe the adversary's queries (via $\mathbf{Q}$), and the other requires the ability to program the RO (via $\mathbf{P}$). We will illustrate the latter. (Note that these techniques are also sometimes applied in the generic group model [58], cf. [22].)

A Schnorr signature [57] of a message M under secret key $s \in \mathbb{Z}_n$ over a finite group $\langle G \rangle$ of order n is a pair (x, t), where $x = r - st \pmod{n}$, $t = \mathbf{R}(\underline{rG} \,\|\, M)$, and r is a random element of $\mathbb{Z}_n$. Valid Schnorr signatures can be simulated without knowledge of s by choosing random $x, t \in \mathbb{Z}_n$, computing $R = xG + tP$, where $P = sG$ is the public key, and setting $\mathbf{P}(\underline{R} \,\|\, M, t)$. The idea is that, in a security reduction for the target cryptosystem, an adversary that simulates the RO can program it in this way without significantly changing the distribution of the RO-query responses. However, when composing the simulator with an adversary in a security reduction, one must take care to ensure that whatever power the simulator is endowed with is also available in the reduction. For example, suppose we find a reduction from a problem—say, decisional Diffie-Hellman (DDH)—to the IND-CCA security of an encryption scheme with key pair (sG, s). If we want to prove that this scheme is secure in the presence of a Schnorr-signature oracle for s, then the DDH adversary we construct must have the ability to program the RO. Indeed, this is often the case, as it is here; the DDH adversary would simply simulate the RO itself. We introduce *oracle-relative* simulators as a means of formalizing the requirements of the simulator for composition.

Definition 6 (Oracle-relative simulators)**.** Let $\mathcal{O}$ be an oracle in an experiment. An *$\mathcal{O}$-relative* simulator $\mathcal{S}$ is one for which both the initiator and operator expect oracle access to $\mathcal{O}$; we say that $\mathcal{S}$ is *c-$\mathcal{O}$-bound* if each algorithm makes at most c such queries on any execution. Let $\mathcal{X}$ and $\mathcal{Y}$ be sets and let $\mu_1, \mu_2 \geq 0$ be real numbers. In the ROM we say that a $\mathbf{P}$-relative simulator is *(μ_1, μ_2)-min-entropy* if for all $(X', Y') \in \mathcal{X} \times \mathcal{Y}$ and each query (X, Y) to $\mathbf{P}$, it holds that $\Pr\big[\, X = X' \,\big] \leq 2^{-\mu_1}$ and $\Pr\big[\, Y = Y' \,\big] \leq 2^{-\mu_2}$. ♦

Theorem 1. *Let $\mathcal{I}^1$ and $\mathcal{I}^0$ be interfaces, let $\mathcal{G}$ be a game, and let $H : \{0,1\}^* \to \{0,1\}^h$ be a function modeled as a random oracle. Let $q_G, q_I, q_R, t, c_I, c_R, c_P, s \geq 0$ be integers such that $s = O(t/(q_I + 1))$, and let $\mu_1, \mu_2 \geq 0$ be real numbers such that $\mu_2 \leq h$. Let $\boldsymbol{r} = (t, q_G, q_I, q_R)$. Then, for every regular, $\mathbf{P}$- and $\mathbf{Q}$-relative simulator $\mathcal{S}$ that is (s, c_I, c_R)-resource, c_P-$\mathbf{P}$-bound, and (μ_1, μ_2)-min-entropy, it holds that*

(i) $\mathbf{Adv}^{\mathrm{sec/i}}_{\mathcal{I}^1, \mathcal{G}}(\boldsymbol{r}) \leq \epsilon + \mathbf{Adv}^{\mathrm{sec}}_{\mathcal{I}^1, \mathcal{G}}(O(t), q_G, \hat{q_R}) + \mathbf{Adv}^{\mathrm{gap1}}_{\mathcal{I}^1, \mathcal{G}}(\mathcal{S}, \hat{\boldsymbol{r}})$ *and*

(ii) $\mathbf{Adv}^{\mathrm{sec/i}}_{\mathcal{I}^1, \mathcal{G}}(\boldsymbol{r}) \leq \epsilon + \mathbf{Adv}^{\mathrm{sec/i}}_{\mathcal{I}^0, \mathcal{G}}(O(t), q_G, c_I q_I, \hat{q_R}) + \mathbf{Adv}^{\mathrm{gap2}}_{\mathcal{I}^1, \mathcal{I}^0, \mathcal{G}}(\mathcal{S}, \hat{\boldsymbol{r}})$,

where $\epsilon = (c_P q_I)(q_R / 2^{\mu_1 - 1} + 2^{h - \mu_2} - 1)$, $\hat{q_R} = q_R + (c_R + c_P)(q_I + 1)$, and $\hat{\boldsymbol{r}} = (O(t), q_G, q_I, \hat{q_R})$.

We give the full proof in Appendix A.1. Except for accounting for the simulator's powers in the ROM, the proof is closely related to [55, Theorem 1]. A few observations about this result are in order. First, we note that the ϵ term in the bound is only non-zero for simulators that program the RO. Second, it is sufficient for the domain points programmed by the simulator to be high min-entropy, but the bound is vacuous unless the corresponding range points are essentially uniform (because of the $2^{h - \mu_2}$ term in the expression for ϵ).

When the programmed domain points are high min-entropy, neither the game nor the GAP2 distinguisher is likely to call the RO on the domain points programmed by the simulator. This fact, and the uniformity of programmed range points, allows us to compose the GAP1/GAP2 distinguisher and the simulator $\mathcal{S}$ into a new SEC/I adversary, despite the fact that $\mathcal{S}$ may program the RO, but the SEC/I adversary may not. Likewise, the simulator "observing" the distinguisher's RO queries is not an issue for this composition.

A necessary condition for Theorem 1(ii). Condition (ii) of the composition theorem characterizes a sufficient property of $(\mathcal{I}^1, \mathcal{I}^0)$ and $\mathcal{G}$ such that it is safe to replace $\mathcal{I}^0$ with $\mathcal{I}^1$ (GAP2). This tells us, in particular, what sorts of operations are safe to expose in an API without breaking applications. We would also like a characterization of what sorts of operations are *not* safe, i.e., a necessary condition for Theorem 1(ii). We find that if wGAP2 security (defined below) does not hold for $(\mathcal{I}^1, \mathcal{I}^0)$, then there are games $\mathcal{G}$ for which $\mathcal{I}^1$ is not SEC/I secure, even if $\mathcal{I}^0$ is SEC/I secure for $\mathcal{G}$ (Theorem 2). We will use this result to rule out certain API-design choices in the remainder of the paper.

Definition 7 (wGAP2 security)**.** The wGAP2 experiment is defined in Figure 3. A wGAP2 adversary takes as input a string and outputs a bit and expects access to an interface oracle. Let $\mathcal{I}^1$ and $\mathcal{I}^0$ be interfaces, $\mathcal{S}$ be a simulator, and $\mathcal{D}$ be a wGAP2 adversary. The wGAP2 experiment for $(\mathcal{I}^1, \mathcal{I}^0)$, $\mathcal{S}$, and $\mathcal{D}$, denoted $\mathbf{Exp}^{\mathrm{wgap2}}_{\mathcal{I}^1,\mathcal{I}^0}(\mathcal{S}, \mathcal{D})$, is defined just like the GAP2 experiment in Figure 3, except that $\mathcal{D}$ is only executed with access to oracle **Op**, and since there is no game context, we remove line 3:20. Define the advantage of $\mathcal{D}$ in distinguishing $\mathcal{I}^1$ from $\mathcal{I}^0$ with respect to simulator $\mathcal{S}$ as $\mathbf{Adv}^{\mathrm{wgap2}}_{\mathcal{I}^1,\mathcal{I}^0}(\mathcal{S}, \mathcal{D}) = 2\Pr\big[\,\mathbf{Exp}^{\mathrm{wgap2}}_{\mathcal{I}^1,\mathcal{I}^0}(\mathcal{S}, \mathcal{D})\,\big] - 1$. Informally, we say that $(\mathcal{I}^1, \mathcal{I}^0)$ is wGAP2 secure if for every efficient adversary $\mathcal{D}$, there is an efficient simulator $\mathcal{S}$ such that $\mathcal{D}$'s advantage is small. We say $\mathcal{D}$ is (t, q_I)-resource if it is t-time and makes at most q_I queries to **Op**. ◆

Theorem 2 (wGAP2 is necessary for Theorem 1(ii))**.** *Let $\mathcal{I}^1$ and $\mathcal{I}^0$ be interfaces, let $\mathcal{B}$ be an SEC/I adversary, and let $\mathcal{D}$ be a wGAP2 adversary. There exist a game $\mathcal{G}$, SEC/I-adversary $\mathcal{A}$, and simulator $\mathcal{S}$ such that*

$$\mathbf{Adv}^{\mathrm{wgap2}}_{\mathcal{I}^1,\mathcal{I}^0}(\mathcal{S}, \mathcal{D}) + \mathbf{Adv}^{\mathrm{sec/i}}_{\mathcal{I}^0,\mathcal{G}}(\mathcal{B}) \le \mathbf{Adv}^{\mathrm{sec/i}}_{\mathcal{I}^1,\mathcal{G}}(\mathcal{A})\,.$$

Moreover, if $\mathcal{D}$ is (s, r)-resource, $\mathcal{B}$ is (t, q_G, q_I)-resource, and $t = O(s)$, then $\mathcal{A}$ is $(O(t), q_G, q_I + r)$-resource and $\mathcal{S}$ is $(t, 1)$-resource.

(Note that this result is easily lifted to the ROM.) The proof (provided in Appendix A.2) is in the same spirit as that of [47, Theorem 1], but there are some subtleties. The crux of the argument, which was adapted from Maurer, Renner, and Holenstein [47], is that the game $\mathcal{G}$ is defined using the adversary $\mathcal{D}$ so that the winning condition depends on $\mathcal{D}$ doing something "bad" (in particular, outputting 1). This allows us to relate $\mathcal{B}$'s advantage to $\mathcal{D}$'s. It may be that GAP2 is, itself, necessary, but we know no way of showing it. In particular, the proof trick we use will not work, because $\mathcal{D}$'s experiment would allow it to call the game $\mathcal{G}$, which in turn would run $\mathcal{D}$, and so on.

Remark 1. We do not know of a necessary condition for Theorem 1(i). The obvious route of defining a "weak version" of GAP1 (in the way that wGAP2 is a weak version of GAP2) and adapting Theorem 2 does not work, since the simulator used in that proof uses its interface oracle in a crucial way. Our sense is that such a necessary condition would depend on the details of the interface. We will leave addressing this apparent asymmetry between GAP1 and GAP2 for future work.

5 Discrete Log Interfaces

We bring our framework to bear on a few common operations for discrete log (DL) interfaces. We first recall some standard definitions from the cryptographic literature and formally define *DL* interfaces and *signing* interfaces.

Preliminaries. Refer to the CDH and GDH experiments in Figure 4. Define the advantage of an adversary $\mathcal{A}$ in solving an instance of the *computational DH (CDH)* problem for $\mathbb{G}$ as $\mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathrm{cdh}}_{\mathbb{G}}(\mathcal{A})]$ and let $\mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(t)$ denote the maximum advantage of any t-time CDH-adversary. Define the advantage of an adversary $\mathcal{A}$ in solving an instance of the *gap DH (GDH)* problem [49] for $\mathbb{G}$ as $\mathbf{Adv}^{\mathrm{gdh}}_{\mathbb{G}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathrm{gdh}}_{\mathbb{G}}(\mathcal{A})]$. Depending on the group $\mathbb{G}$ and the model of computation, it may not be possible to evaluate $\mathcal{A}$'s **DDH**

```
$\mathbf{Exp}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\mathcal{A})$
1  $\mathbf{dec}\ X, Z \in \mathbb{G};\ y \in \mathbb{Z}_n$
2  $(\underline{X}, sk) \twoheadleftarrow \mathcal{I}.\mathrm{Gen}(\,)$
3  $y \twoheadleftarrow \mathbb{Z}_n$
4  $Z \twoheadleftarrow \mathcal{A}^{\mathcal{I}_{sk}}(X, yG)$
5  $\mathbf{ret}\ (Z = yX)$
```

```
$\mathbf{Exp}^{\mathrm{cdh}}_{\mathbb{G}}(\mathcal{A})$ / $\boxed{\mathbf{Exp}^{\mathrm{gdh}}_{\mathbb{G}}(\mathcal{A})}$
6  $x, y \twoheadleftarrow \mathbb{Z}_n$
7  $Z \twoheadleftarrow \mathcal{A}(xG, yG)$
8  $\boxed{Z \twoheadleftarrow \mathcal{A}^{\mathbf{DDH}}(xG, yG)}$
9  $\mathbf{ret}\ (Z = xyG)$
```

```
$\mathbf{DDH}(A, B, C)$
10  $a \leftarrow \log_G A$
11  $b \leftarrow \log_G B$
12  $c \leftarrow \log_G C$
13  $\mathbf{ret}\ (c = ab)$
```

Figure 4: Let $\mathbb{G} = \langle G \rangle$ be a represented, additive group of order n and let $\mathcal{I}$ be a DL interface for $\mathbb{G}$. Left: IDH problem for $(\mathbb{G}, \mathcal{I})$. Right: CDH and GDH problems for $\mathbb{G}$.

---

queries efficiently; for the purpose of accounting for $\mathcal{A}$'s resources, we will regard the discrete log computations on lines 4:7–8 as constant time operations. Let $\mathbf{Adv}^{\mathrm{gdh}}_{\mathbb{G}}(t, q)$ denote the maximum advantage of any t-time GDH-adversary that makes at most q queries to its **DDH** oracle. Informally, we say CDH (resp. CDH) is hard for $\mathbb{G}$ if the CDH (resp. GDH) advantage of any efficient adversary is small. Define the CR advantage of an adversary $\mathcal{C}(\,) \mapsto \mathbf{elem}_{\mathcal{X}}, \mathbf{elem}_{\mathcal{Y}}$ in finding collisions for function $H : \mathcal{X} \to \mathcal{Y}$ as $\mathbf{Adv}^{\mathrm{cr}}_{H}(\mathcal{C}) = \Pr\big[\, X \neq Y \wedge H(X) = H(Y) : (X, Y) \twoheadleftarrow \mathcal{C} \,\big]$.

Definition 8 (DL and signing interfaces)**.** Let $\mathbb{G} = \langle G \rangle$ be a represented, additive group of order n. A DL interface for $\mathbb{G}$ is an interface $\mathcal{I}$ with an associated *scalar computer*, a deterministic algorithm $\mathrm{Scal}(\mathbf{str}\ sk) \mapsto \mathbf{int}\ s$ such that for every $(pk, sk) \in [\mathcal{I}.\mathrm{Gen}(\,)]$ it holds that $pk = \underline{sG}$, where $s = \mathcal{I}.\mathrm{Scal}(sk)$. We say that $\mathcal{I}$ is *simple* if $\mathcal{I}.\mathrm{Scal}(sk) = s$ just in case $sk = \underline{s}$.

A *signing interface* $\mathcal{DS}$ has an associated deterministic algorithm $\mathcal{DS}.\mathrm{Verify}(\mathbf{str}\ pk, ctx, M, T) \mapsto \mathbf{bool}\ v$, called the *verifier*, for which $T \in [\mathcal{DS}(sk, ctx, \mathsf{sig}, M)]$ iff $\mathcal{DS}.\mathrm{Verify}(pk, ctx, M, T) = 1$ for all $ctx, M, T \in \{0,1\}^*$ and $(pk, sk) \in [\mathcal{DS}.\mathrm{Gen}(\,)]$. (This is analogous to the correctness condition for standard signature schemes.) We may denote $\mathcal{DS}(sk, ctx, \mathsf{sig}, M)$ by $\mathcal{DS}.\mathrm{Sign}(sk, ctx, M)$ and refer to $\mathcal{DS}.\mathrm{Sign}$ as the *signer*. We say that a game is $\mathcal{DS}$*-regular* (for the associated experiment) if each time it invokes $\mathcal{DS}.\mathrm{Verify}$, it does so on input of (pk, α, M, T), where α is the game context used to initialize it and $pk, M, T \in \{0,1\}^*$. ◆

5.1 Diffie-Hellman

Let $\mathbb{G} = \langle G \rangle$ be an additive, represented group of order n. Let $\mathcal{I}$ be a DL interface for $\mathbb{G}$ and define $\mathcal{I}_{+\mathrm{dh}}$ as the pair of algorithms $(\mathcal{I}.\mathrm{Gen}, \mathrm{Op})$, where Op is defined as follows. On input of (sk, ctx, op, in), if $op = \mathsf{dh}$, then it interprets in as $\underline{Q}$, where $Q \in \mathbb{G} \cup \{\diamond\}$. If $Q \neq \diamond$, then it returns $\underline{sQ}$, where $s = \mathcal{I}.\mathrm{Scal}(sk)$; otherwise it returns $\mathcal{I}(sk, ctx, op, in)$. We refer to dh as the *DH operator*. (Note that point validation [45] for this operation is implicitly enforced by our conventions for represented groups; see Section 2.)

With the help of such a "static DH oracle", an algorithm devised by Brown and Gallant [19] significantly reduces the cost of computing discrete logarithms in many finite groups. Given a point $P = sG \in \mathbb{G}$ and an oracle that computes sQ for a chosen input $Q \in \mathbb{G}$, their algorithm correctly computes s in $O(n^{1/3})$ time, where n is the order of $\mathbb{G}$. This is a significant improvement over the $O(n^{1/2})$ complexity of the best known classical algorithm for solving the discrete log problem [53]. But in order to be able to finish the computation in this amount of time, their algorithm needs to perform about $n^{1/3}$ queries; thus, choosing a large enough group may render a key-recovery attack infeasible in practice. However, we can rule out the security of exposing the DH operation (inadvertently or not) in a given interface as follows. We formalize a property of $\mathcal{I}$ that, if it holds, implies that $(\mathcal{I}_{+\mathrm{dh}}, \mathcal{I})$ is wGAP2 *insecure*; by Theorem 2, this implies that $\mathcal{I}_{+\mathrm{dh}}$ is not SEC/I secure in general. We then build on this result by considering whether it is safe to expose some function of the output (e.g., a hash or key-derivation function); when we (conservatively) model the function as a random oracle, we find that this is not wGAP2 secure.

Insecurity of exposing DH easily follows from the hardness of a variant of the CDH problem for $\mathbb{G}$ associated with $\mathcal{I}$. The problem is motivated by the strong DH (SDH) problem proposed by Abdalla, Bellare, and Rogaway [1]. The SDH problem is similar to CDH, except that in addition to $xG, Y \in \mathbb{G}$, the adversary is given an oracle that, on input of (P, Q), returns true iff $Q = xP$. This is a kind of "restricted" DDH oracle whereby one of the inputs (xG) is fixed. The *interface-relative DH (IDH)* problem for $(\mathbb{G}, \mathcal{I})$ is as follows.

Definition 9 (The IDH problem)**.** Refer to the IDH experiment for $\mathbb{G}$ and $\mathcal{I}$ in Figure 4. The experiment first runs $\mathcal{I}$.Gen to get the public key X and secret key sk. It then chooses a random $y \in \mathbb{Z}_n$ and runs the adversary $\mathcal{A}$ on input of (X, yG) and with oracle access to $\mathcal{I}_{sk}$; the adversary wins if it outputs yX. Define the advantage of IDH-adversary $\mathcal{A}$ as $\mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\mathcal{A}) = \Pr\big[\,\mathbf{Exp}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\mathcal{A})\,\big]$. An IDH adversary is (t, q)-resource if it is t-tim4 and makes at most q queries to its interface oracle; we denote the maximum advantage of any $\boldsymbol{r}$-resource IDH adversary by $\mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\boldsymbol{r})$. Informally, we say the IDH problem is hard for $(\mathbb{G}, \mathcal{I})$ if $\mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\mathcal{A})$ is small for every efficient $\mathcal{A}$. ◆

We will use this problem as a sort of litmus test to rule out insecure API designs. In Section 5.2 we show (via Theorem 1(ii)) that CDH and IDH are equivalent relative to EdDSA, and in in Section 6 we show that GDH and IDH are equivalent relative to Noise. To prove that hardness of the IDH problem for $(\mathbb{G}, \mathcal{I})$ implies the wGAP2 insecurity of $(\mathcal{I}_{+\mathrm{dh}}, \mathcal{I})$, we exhibit a wGAP2 adversary $\mathcal{D}$ such that in order for any simulator $\mathcal{S}$ to thwart $\mathcal{D}$, it must solve an instance of IDH for $(\mathbb{G}, \mathcal{I})$.

Theorem 3. *Suppose that n is prime and let $t, q_I \geq 0$ be integers. There is a $(O(t), 1)$-resource wGAP2-adversary $\mathcal{D}$ such that for all (t, q_I)-resource $\mathcal{S}$, there is a $(O(t), q_I)$-resource IDH-adversary $\mathcal{A}$ such that* $\mathbf{Adv}^{\mathrm{wgap2}}_{\mathcal{I}_{+\mathrm{dh}},\mathcal{I}}(\mathcal{S}, \mathcal{D}) = 1 - \mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\mathcal{A})$.

Proof. Define adversary $\mathcal{D}^{\mathbf{Op}}(\underline{P})$ as follows. First run $r \twoheadleftarrow \mathbb{Z}_n$, then ask $\underline{Z} \twoheadleftarrow \mathbf{Op}(\varepsilon, \mathsf{dh}, \underline{rG})$. If $r^{-1}Z = P$, then return 1; otherwise return 0. Let d_{b1} denote the probability that $\mathcal{D}$ outputs 1 conditioned on the event that its challenge bit is b. First, if $b = 1$, then the response to $\mathcal{D}$'s query will be $Z = srG$, where $P = sG$. Since n is prime, r has a unique inverse $1/r \pmod{n}$, and so $r^{-1}Z = r^{-1}srG = sG = P$. It follows that $d_{11} = 1$. Now consider the probability that $r^{-1}Z = P$ given that $b = 0$ and define adversary $\mathcal{A}^{\mathcal{O}}(P, Q)$ as follows. It first executes $\sigma \twoheadleftarrow \mathcal{S}.\mathrm{Init}(\underline{P})$, then $\underline{Z} \twoheadleftarrow \mathcal{S}^{\mathcal{O}}(\&\sigma, \varepsilon, \mathsf{dh}, \underline{Q})$. Finally, it returns Z. Then the probability that $\mathcal{A}$ wins is precisely the probability that, in $\mathcal{D}$'s game, simulator $\mathcal{S}$ outputs $\underline{Z}$ such that $r^{-1}Z = P \iff rP = Z$, and so $d_{01} = \Pr\big[\,\mathbf{Exp}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\mathcal{A})\,\big]$. ☐

Functional DH. Many applications do not make direct use of static DH, but some function of its output. In particular, it is common to apply a hash or key-derivation function to the shared secret, perhaps binding it to some context, e.g., the transcript hash in TLS or, as we will see, the CipherState in Noise. Therefore, it is worth considering whether exposing this intermediate functionality is secure.

Let $\mathcal{F} : \mathbb{G} \times \{0,1\}^* \to \{0,1\}^h$ be a function. Define the interface $\mathcal{I}_{+\mathrm{fdh}}$ as the pair of algorithms $(\mathcal{I}.\mathrm{Gen}, \mathrm{Op})$, where Op is defined as follows. On input of (sk, ctx, op, in), if $op = \mathsf{fdh}$, then interpret in as $\underline{Q}$, where Q is an element of $\mathbb{G} \cup \{\diamond\}$. If $Q \neq \diamond$, then return $\mathcal{F}(sQ, ctx)$, where $s = \mathcal{I}.\mathrm{Scal}(sk)$; otherwise return $\mathcal{I}.\mathrm{Op}(sk, ctx, op, in)$. We call $op = \mathsf{fdh}$ the *functional DH operator*.

Exposing functional DH is also wGAP2 insecure. The proof is more involved, but follows similar lines as Theorem 3. We cannot directly exploit the algebraic structure of the DH operator as we did above, since rather than getting $\underline{sQ}$ in response to its query, adversary $\mathcal{D}$ gets $\mathcal{F}(sQ, ctx)$. Instead, we model $\mathcal{F}$ as a random oracle and hope that the simulator manages to query the oracle with the correct point. We prove the following in Appendix A.3:

Theorem 4. *Suppose that n is prime and let $t, q_I, q_R \geq 0$ be integers. When $\mathcal{F}$ is modeled as a random oracle, there is a $(O(t), 1, 1)$-resource wGAP2-adversary $\mathcal{D}$ such that for all (t, q_I, q_R)-resource,* $\mathbf{P}$*- and* $\mathbf{Q}$*-relative, and p-*$\mathbf{P}$*-bound $\mathcal{S}$, there is a $(O(t + q_R), q_I)$-resource IDH-adversary $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\mathrm{wgap2}}_{\mathcal{I}_{+\mathrm{fdh}},\mathcal{I}}(\mathcal{S}, \mathcal{D}) + \epsilon \geq 1 - \mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\mathcal{A})\,,$$

where $\mathcal{I}$ is 0-ro-bound, $\epsilon = \hat{q}/n + \hat{q}^2/2^{h-1}$, and $\hat{q} = 2(q + p)$.

Discussion. The existence of a static DH oracle in an interface can be difficult to recognize, and its impact on security is often quite subtle. Acar, Ngyuen, and Zavarucha [3] discovered that an early version of the TPM standard exposed such an oracle via flexible API calls designed to support a wide variety of protocols. Indeed, a rigorous analysis of the standard in our attack model would have unearthed this subtlety. It would be worthwhile to study the proposal of Camenish et al. [20], which aims to remove the TPM oracle while still supporting a large variety of useful applications. More generally, we suggest that the approach developed in this paper could be used to vet API standards before they are implemented to help uncover such flaws. Though the problem with TPM was obvious in hindsight, it is possible that more flaws lurk in this and other API designs.

```
Gen( )
    $K \twoheadleftarrow \{0,1\}^b$; $s \leftarrow \mathrm{Scal}(K)$
    ret $(\underline{sG}, K)$

Verify($pk, ctx, M, T$)
    dec $P, R \in \mathbb{G}$; $x, t \in \mathbb{N}$
    $\underline{P} \leftarrow pk$; $\underline{R, x} \leftarrow T$
    if $\neg R \vee \neg x$ then ret 0
    $\underline{t}_{2b} \leftarrow \mathcal{H}(vr(ctx) \,\|\, \underline{R} \,\|\, \underline{P} \,\|\, ph(M))$
    ret $\big(x2^cG = 2^cR + t2^cP\big)$

Scal($K$)
    ret $cl(\mathcal{H}(K)[:b])$

Sign($K, ctx, M$)
    dec $r, t \in \mathbb{N}$
    $s \leftarrow \mathrm{Scal}(K)$; $X \leftarrow \mathcal{H}(K)[b+1:]$
    $\underline{r}_{2b} \leftarrow \mathcal{H}(vr(ctx) \,\|\, X \,\|\, ph(M))$
    $\underline{t}_{2b} \leftarrow \mathcal{H}(vr(ctx) \,\|\, \underline{rG} \,\|\, \underline{sG} \,\|\, ph(M))$
    $x \leftarrow r + st \pmod{n}$
    ret $\underline{rG, x}$
```

Figure 5: Signing/DL interface $\mathcal{ED}$ for EdDSA. Let $b, c \in \mathbb{N}$ and let $\mathbb{G} = \langle G \rangle$ be a represented, additive group of order n. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{2b}$, $cl : \{0,1\}^b \to \mathbb{Z}_n$, and $vr, ph : \{0,1\}^* \to \{0,1\}^*$ be functions.

5.2 EdDSA

Unlike signature schemes like RSA-PSS or ECDSA, the standardized version of EdDSA (RFC 8032 [35]) admits variants that are context separable, allowing us to prove it GAP1 secure (in the ROM) for any game in which all signing and verifying operations are regular (Def. 8). We also show that any variant can be securely composed with any simple DL interface. After presenting our results, we will make the case for designing and deploying context-separable signatures in practice.

The standard specifies two concrete instantiations of EdDSA: Ed22519 and Ed448, whose names indicate the underlying group. The signing interface $\mathcal{ED}$ defined in Figure 5 specifies generic EdDSA; a concrete scheme is instantiated by selecting the group $\mathbb{G}$, integers b and c, and functions $\mathcal{H}$, cl, vr, and ph. The group is determined by a prime number $p > 2$, parameters for a (twisted) Edwards curve E (see [14, Section 2]), and a generator G of a prime order subgroup of $E(\mathbb{F}_p)$, where $E(\mathbb{F}_p)$ denotes the group of points $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ that lie on the curve E, and $\mathbb{F}_p$ denotes the finite field of order p. Define b so that $2^{b-1} > p$ and define c so that $\#E(\mathbb{F}_p) = n2^c$ (i.e., 2^c is the cofactor of $\mathbb{G}$). This choice of b makes it possible to encode signatures with $2b$ bits, and this choice of c is intended to mitigate small subgroup attacks [45]. The "clamping" function cl is similarly tailored to the underlying group: for Ed25519 and its variants, this function clears the first 3 bits, sets the second to last bit, and clears the last bit. Finally, the algorithm variant is determined by the functions vr and ph. For example, the most common Ed25519 variant is obtained by setting $vr(X) = \varepsilon$ and $ph(X) = X$ for all X, but the standard also specifies variants that permit context (Ed25519ctx) and pre-hashing of the message (Ed25519ph). To provide context separability, the function vr must be collision resistant.

We begin our analysis by proving that the context-separable variants of EdDSA are GAP1 secure in the ROM for games in which the signing and verifying operations are regular. Our bound implies (via Theorem 1(i)), that the CDH and IDH problems are equivalent for *all* variants of EdDSA. (This means, in particular, that exposing (functional) DH for a signing EdDSA key is not secure in general.) We will then show that exposing *any* variant of EdDSA in any *simple* DL interface is GAP2 secure in general. Fix EdDSA parameters $(\mathbb{G}, \mathcal{H}, cl, vr, ph, b, c)$ and let $\mathcal{ED}$ be the signing interface instantiated with these parameters as specified in Figure 5. Let $n = |\mathbb{G}|$.

Theorem 5. *Let $\mathcal{G}$ be an ($\mathcal{ED}$-)regular game and suppose that $n \leq 2^{b-1}$. When $\mathcal{H}$ is modeled as a random oracle, there exists a regular, $\mathbf{P}$-relative simulator $\mathcal{S}$ such that for all $t, q_G, q_I, q_R, c \geq 0$ there exists a $O(t + q_R q_I)$-time CR-adversary $\mathcal{C}$ such that $\mathbf{Adv}^{\mathrm{gap1}}_{\mathcal{ED},\mathcal{G}}(\mathcal{S}, \boldsymbol{r}) \leq 2cq_I \mathbf{Adv}^{\mathrm{cr}}_{vr}(\mathcal{C}) + 6q_R q_I/n$, where $\mathcal{G}$ is c-bound, $\mathcal{S}$ is $(\log n/2, 2b)$-min-entropy, $(O(t/(q_I + 1)), 1, 0)$-resource, and 1-$\mathbf{P}$-bound, and $\boldsymbol{r} = (t, q_G, q_I, q_R)$.*

Refer to Appendix A.4 for the full proof. The simulator programs the random oracle with valid EdDSA signatures much like one would for Schnorr signatures. (See the discussion in Section 4.2.) We must ensure, however, that signatures programmed by the simulator cannot be used by the adversary in an attack against the game $\mathcal{G}$. To do so, we use the collision resistance of vr to bound the probability that any interface query made via **Call** coincides with an interface query made via **Op**. For this argument to work, we must require that $\mathcal{G}$ is ($\mathcal{ED}$-)regular.

If the game in Theorem 5 makes no interface queries (i.e., is 0-bound), then CR security of vr is not required. This allows us to prove equivalence of IDH and CDH regardless of how vr is realized. The following corollary follows almost immediately from Theorems 1(i) and 5.

Corollary 1. *Let $r = |\mathrm{Rng}\, cl|$ and suppose that $r \mid 2^b$ and $n \le 2^{b-1}$. Then for all $t, q_I, q_R > 0$ it holds that $\mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{ED}}(t, q_I, q_R) \le n/r\mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(O(t + \hat{q})) + 7q_Rq_I/n$, where $\mathcal{H}$ is modeled as a random oracle and $\hat{q} = q_R + q_I + 1$.*

Proof. Let $\mathcal{A}$ be a (t, q_I, q_R)-resource IDH adversary and consider the following game $\mathcal{G}^{\mathrm{cdh}}$: on input of $(\underline{P}, ctx)$, the initiator ignores ctx, samples $y \twoheadleftarrow \mathbb{Z}_n$, and returns $(\underline{P, yP, yG})$; the caller simply returns $\bot$ in response to any query; and on input of $(\underline{P, Z, Z^*})$, the finalizer returns $(Z = Z^*)$. Now consider the following $(O(t), 0, q_I, q_R)$-resource SEC/I-adversary $\mathcal{A}'$: On input of $\underline{P}$, adversary $\mathcal{A}'$ queries $\underline{Q} \twoheadleftarrow \mathbf{Init}(\varepsilon)$, runs $Z^* \twoheadleftarrow \mathcal{A}^{\mathbf{Op}}(P, Q)$, queries $\mathbf{Final}(\underline{Z^*})$, then halts. It is clear that $\mathcal{A}'$ perfectly emulates $\mathcal{A}$ in the IDH experiment relative to $\mathcal{ED}$, and so $\mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{ED}}(\mathcal{A}) = \mathbf{Adv}^{\mathrm{sec/i}}_{\mathcal{ED},\mathcal{G}^{\mathrm{cdh}}}(\mathcal{A}')$.

By Theorem 1, Theorem 5, and the assumption that $n \le 2^{b-1}$, there exists a $(O(t), 0, 0, q + q_I + 1)$-resource SEC/I-adversary $\mathcal{B}'$ such that $\mathbf{Adv}^{\mathrm{sec/i}}_{\mathcal{ED},\mathcal{G}^{\mathrm{cdh}}}(\mathcal{A}') \le 7q_Rq_I/n + \mathbf{Adv}^{\mathrm{sec}}_{\mathcal{ED},\mathcal{G}^{\mathrm{cdh}}}(\mathcal{B}')$. In the remainder, we relate the advantage of $\mathcal{B}'$ to the hardness of the CDH problem for $\mathbb{G}$.

Observe that $\mathcal{B}$'s experiment only uses $\mathcal{ED}$ for key generation, since neither $\mathcal{G}^{\mathrm{cdh}}$ nor $\mathcal{B}'$ makes use of the operator. Consider the following CDH adversary $\mathcal{B}$. On input of (P, Q), it first executes $Z \twoheadleftarrow \langle \mathcal{B}' : \mathbf{Init}', \mathbf{Final}', \bot, \bot, \mathbf{R} \rangle(\underline{P})$, where the oracles are defined as follows: oracle $\mathbf{R}$ is its own RO; on input of ctx, oracle $\mathbf{Init}'$ returns $\underline{Q}$; and on input of $\underline{Z^*}$ to oracle $\mathbf{Final}'$, adversary $\mathcal{B}$ halts and outputs Z^*.

To complete the proof, we argue that $\mathcal{B}$ perfectly emulates $\mathcal{B}'$ in its experiment if the scalar s corresponding to its input P happens to fall in the range of cl. In $\mathcal{B}'$'s experiment, the scalar s is equal to $cl(\mathbf{R}(K)[:b])$. Since $\pi[K]$ is determined by the RO query made by the key generator (the key generator is the first algorithm in the experiment to be executed with access to the RO), we may treat $\mathbf{R}(K)[:b]$ as a random variable V uniformly distributed over $\{0,1\}^b$. Because r divides 2^b, function cl partitions its domain into r sets of equal size. It follows that $cl(V)$ is uniformly distributed over $\mathrm{Rng}\, cl$. Conditioning on the event that $s \in \mathrm{Rng}\, cl$ in $\mathcal{B}$'s experiment, we conclude that $\mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(\mathcal{B}) \ge r/n \Pr\left[\, \mathbf{Exp}^{\mathrm{sec/i}}_{\mathcal{ED},\mathcal{G}^{\mathrm{cdh}}}(\mathcal{B}') \,\right]$. ☐

Remark 2. For both Ed25519 and Ed448 the order of the main subgroup is less than 2^{b-1} [35]. Moreover, function cl is specified so that $|\mathrm{Rng}\, cl| = 2^a$ for some $a < b$, and so the number of distinct points in the range divides 2^b. The multiplicative factor $n/2^a$ is small for both algorithms. Function vr is only CR secure for Ed25519ctx and Ed448ctx, and only so for a finite set of inputs: vr is injective for these variants, but the standard imposes a maximum length on the context string. Therefore, to apply Theorem 5 directly to the standard, it is necessary to limit the adversary's queries in kind.

Finally, we show that EdDSA can be composed with any *simple* DL interface $\mathcal{I}$ without affecting the security of $\mathcal{I}$'s intended application. Let $\mathcal{I}$ be a simple DL interface for $\mathbb{G}$. We define a new interface $\mathcal{ED}_{+\mathcal{I}} = (\mathcal{ED}.\mathrm{Gen}, \mathrm{Op})$, where on input of (sk, ctx, op, in), algorithm Op returns $\mathcal{ED}.\mathrm{Sign}(sk, ctx, in)$ if $op = \textsf{sig}$ and returns $\mathcal{I}.\mathrm{Op}(\underline{s}, ctx, op, in)$ otherwise, where $s = \mathcal{ED}.\mathrm{Scal}(sk)$.

Theorem 6. *Let $\mathcal{G}$ be a game and suppose that $n \le 2^{b-1}$. When $\mathcal{H}$ is modeled as a random oracle, there exists a regular, $\mathbf{P}$-relative simulator $\mathcal{S}$ such that for all $t, q_G, q_I, q_R \ge 0$ it holds that $\mathbf{Adv}^{\mathrm{gap2}}_{\mathcal{ED}_{+\mathcal{I}},\mathcal{I},\mathcal{G}}(\mathcal{S}, \boldsymbol{r}) \le 7q_Rq_I/n$, where $\mathcal{S}$ is $(\log n/2, 2b)$-min-entropy, $(O(t/(q_I+1)), 1, 0)$-resource, and 1-$\mathbf{P}$-bound, and $\boldsymbol{r} = (t, q_G, q_I, q_R)$.*

The restriction to simple interfaces is so that we can achieve context separation in the proof without using collision resistance of vr. The argument leverages the fact that $\mathcal{I}$ does not make use of the string X computed by the signer. Otherwise the proof is closely related to Theorem 5; we defer the details to to Appendix A.5.

Remark 3. While most signatures are not GAP1 secure (context separability is not a common design pattern in signature schemes), it is likely that exposing many of them in an interface is GAP2 secure for large classes of games. Haber and Pinkas [34] exhibit a programming simulator for RSA-PSS that is high min-entropy in the sense required by Theorem 1. Degabriele et al. [22] show how to simulate ECDSA signatures in the generic group model; in the ROM, techniques of Fersch, Kiltz, and Poettering [29] may be applicable to ECDSA.

Discussion. The restrictions imposed on the game in Theorem 5 and the interface in Theorem 6 are very mild, but are required for enforcing context separation. If the game encodes the UF-CMA security of $\mathcal{ED}$, then this ensures that a signature generated via the interface cannot be used as a forgery in the game. But this "attack" is rather uninteresting and is only an artifact of our model. On the other hand, the game might specify the use of a signature scheme in a complex protocol like TLS in which digital signatures have a variety of uses, including client and server authentication and delegation of credentials for terminating TLS on a party's behalf [7]. In each of these cases the protocol binds the signature to a unique context string identifying its use (e.g., [54, Section 4.4.3]). Our abstraction boundary makes the requirements for such applications explicit. Because Ed25519ctx and Ed448ctx are context separable, Theorem 5 makes clear the conditions under which these algorithms are secure for their intended application, no matter how else they are used: the implementer must ensure that (1) the interface enforces context separation, and (2) signing/verification operations in the application always use the context that identify the application. We believe that exploiting this property of context-separable signatures would reduce the inherent complexity of designing and deploying protocols. (Indeed, it is also not difficult to design signature schemes to have this property.)

6 Noise

In this section we consider the GAP1 security of Noise [52], a framework for designing DL-based, two-party protocols. Noise provides a set of rules for processing *handshake patterns*, which define the sequence of interactions between an *initiator* and *responder* in a protocol. The processing rules involve three primitives: Diffie-Hellman (DH), an AEAD scheme, and a hash function. Each message sent or received by a host updates the host's state, which consists of the host's *ephemeral* (i.e., short-lived) and *static* (long-lived) secret keys, the peer's ephemeral and static public keys, shared state used to derive the symmetric key and associated data, the current symmetric key, and the current nonce. The symmetric key, nonce, and associated data are used to encrypt *payloads* accompanying each message, providing implicit authentication of a peer via confirmation of knowledge of their static secret.

Noise admits a wide variety of protocols. The processing rules are designed to make it easy to verify properites of handshake patterns, and considerable effort has gone into their formal analysis [25, 37, 46]. But the study of handshake patterns in isolation does not fully address the complexity of using Noise to build and deploy protocols. In practice, it is often necessary for the communicants to negotiate the details of the handshake, including the pattern, primitives, and cryptographic artifacts such as static keys and their certificates. All of this is out of scope of the core Noise specification, which aims to be as rigid as possible. As a result, there is an apparent gap between our understanding of the security that Noise provides and how it is used in practice. One question that arises, which we will address here, is whether it is safe to reuse a single static key in many patterns.

We cast the Noise framework as an interface that exposes a host's static key for use in Noise protocols. The interface specifies how the host consumes (resp. produces) messages sent by (resp. to send to) the peer, and how its handshake state is updated as a side-effect. In other words, it implements the processing rules such that Noise patterns can be executed by making calls to the interface. Our goal is to prove GAP1 security with respect to the largest possible set of games, which would provide two benefits in practice. First and foremeost, it would imply joint security (up to context separation) of all patterns the interface implements; second it would provide a degree of robustness to cross protocol attacks by ensuring that, as long as context separation is enforced, vulnerabilities in one application cannot creep into another.

Our analysis sheds light on two limitations of Noise with respect to our security notions. The first is that *some* handshake patterns, if implemented by our interface, would allow for GAP1 attacks. We provide a formal characterization of the actions that give rise to these attacks, and we prove GAP1 security of our interface when they are excluded. The second issue is more subtle. To prove GAP1 security with respect to games in which the adversary may compromise the handshake state—for example, when modeling forward secrecy—it is necessary to tweak the Noise spec slightly. The processing rules explicitly bind the protocol context (i.e., a string that uniquely defines the handshake pattern and parameters) to the initial state of the protocol. While this provides a certain degree of context separability, the lack of binding to each state update precludes a proof of security relative to such games. We propose a simple and efficient modification

```
$\mathbf{Exp}^{\text{int-ctxt}}_{\mathcal{AE}}(\mathcal{A})$
  dec set $\mathcal{Q}, \mathcal{C}$; str $K$; bool $win$
  $K \twoheadleftarrow \mathcal{K}$; $\mathcal{A}^{\mathbf{Enc},\mathbf{Dec}}$; ret $win$

$\mathbf{Enc}(N, A, M)$
  if $N \in \mathcal{Q}$ then ret $\perp$
  $C \leftarrow \mathcal{AE}.\mathrm{Enc}^{N,A}_{K}(M)$; $\mathcal{C} \leftarrow \mathcal{C} \cup \{(N, A, C)\}$
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{N\}$; ret $C$

$\mathbf{Dec}(N, A, C)$
  if $(N, A, C) \in \mathcal{C}$ then ret $\perp$
  $M \leftarrow \mathcal{AE}.\mathrm{Dec}^{N,A}_{K}(C)$
  $win \leftarrow win \vee (M \neq \perp)$
  ret $M$
```

NN:	NK:	NX:	IKpsk2:
→ e, ← e, ee	← s . . . → e, es ← e, ee	→ e ← e, ee, s, es	← s . . . → e, es, s, ss ← e, ee, se, psk

Figure 6: Top: INT-CTXT for AEAD scheme $\mathcal{AE}$ with key space $\mathcal{K}$. Bottom: Examples of Noise handshake patterns.

of the processing rules that ensures context separability under these conditions, allowing us to prove security under minimal (and natural) assumptions about the game.

Of course, a consequence of these restrictions is that our analysis leaves open the security of key-reuse in Noise *as it is*. At the end of this section, we will discuss what our results mean for Noise in practice and suggest directions for future work.

Preliminaries. Our analysis will use the standard notion of ciphertext integrity of AEAD schemes. A scheme for *authenticated encryption with associated data (AEAD)* is a pair of deterministic algorithms $\mathcal{AE} = (\mathrm{Enc}, \mathrm{Dec})$. The first, $\mathrm{Enc}(\mathbf{str}\, K, N, A, M) \mapsto \mathbf{str}\, C$, maps a key K, nonce N, associated data A, and plaintext M to a ciphertext C. The second, $\mathrm{Dec}(\mathbf{str}\, K, N, A, C) \mapsto \mathbf{str}\, M$, maps K, N, A, and C to M. We respectively define the key, nonce, associated-data (AD), and message space as the sets $\mathcal{K}, \mathcal{N}, \mathcal{A}, \mathcal{M} \subseteq \{0,1\}^*$ for which $\mathrm{Enc}(K, N, A, M) \neq \perp$ if and only if $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$; correctness requires that $\mathrm{Dec}(K, K, N, A, \mathrm{Enc}(K, N, A, M)) = M$ for every such (K, N, A, M). (This condition implies that $\mathcal{AE}$ is both *correct* and *tidy* in the sense of Namprempre, Rogaway, and Shrimpton [48].) We say that $\mathcal{AE}$ has key-length k if $\mathcal{K} = \{0,1\}^k$ and nonce-length n if $\mathcal{N} = \{0,1\}^n$. We will use the standard notion of ciphertext integrity (INT-CTXT) for AEAD schemes in the presence of nonce-respecting adversaries; refer to Figure 6 for its precise definition. Define the advantage of an adversary $\mathcal{A}$ in breaking the ciphertext integrity of $\mathcal{AE}$ as $\mathbf{Adv}^{\text{int-ctxt}}_{\mathcal{AE}}(\mathcal{A}) = \Pr\big[\, \mathbf{Exp}^{\text{int-ctxt}}_{\mathcal{AE}}(\mathcal{A}) \,\big]$. Let $\mathbf{Adv}^{\text{int-ctxt}}_{\mathcal{AE}}(t, q_E, q_D)$ denote the maximum advantage of any t-time adversary making at most q_E (resp. q_D) queries to **Enc** (resp. **Dec**).

6.1 Handshake and message patterns

By way of eliciting the formal tools we will need in our analysis, we begin this section with a brief overview of how handshake patterns are specified. Figure 6 recalls four patterns from the standard [52]. The first, referred to as the "NN" pattern, encodes an unauthenticated DH key exchange as a sequence of *messages*, which in turn encode sequences of *tokens*. In the first message (→ e) the initiator generates an ephemeral DH key pair and sends the public key to the responder. In the next message (← e, ee), the responder generates an ephemeral key pair (e), computes the DH shared secret and derives a symmetric key (ee), then sends the ephemeral public key in its response. Every message includes a possibly AEAD-encrypted *payload*. Encryption is opportunistic. Once a shared secret is established, everything that can be encrypted will be encrypted; if the caller does not provide a payload, then the payload is the empty string.

The NK pattern is a variant of NN that provides authentication of the responder. The main difference is an additional message preceding the ellipses (← s) indicating that the responder's static public key is known to the initiator before the protocol begins. In its first action, the initiator computes the shared secret between this and its ephemeral secret (es) and uses it to encrypt the message payload. This has two effects: first, the initiator proves knowledge of the shared secret to the responder; and second, the responder authenticates itself by proving knowledge of the shared secret to the initiator. These properties are due to the sequence of actions induced by the pattern; if decryption fails, then this indicates that the sender does not know the

```
Gen()
  $K \twoheadleftarrow \{0,1\}^b$
  $s \gets cl(K)$
  ret $(\underline{sG}, \underline{s})$

  dec struct {
    str $P, E, S$ } msg
  dec struct { str $id, psk$;
    int $seq$; str $K, N$;  // CipherState
    str $L, A$;  // SymmetricState
    $Q, R \in \mathbb{G}$; $e \in \mathbb{Z}_n$ } st

Op($sk, ctx, op, in$)
  dec st $hs$; msg $req$; bool $f, r$; str $u, pat, err$
  $s \gets \mathrm{Scal}(sk)$; $\underline{u, f, r, pat} \gets op$; $\underline{hs, in} \gets in$
  if $u \neq$ noise $\vee\ |hs.L| \neq h \vee hs.id \neq vr(ctx)$
    then ret $\bot$
  if $f$ then  // outbound payload
    $(resp, err) \twoheadleftarrow \mathrm{Write}(\&hs, s, r, pat, in)$
    if $\neg err$ then ret $\underline{hs, resp, \diamond}$
  else $\underline{req} \gets in$  // inbound message
    $(out, err) \twoheadleftarrow \mathrm{Read}(\&hs, s, r, pat, req)$
    if $\neg err$ then ret $\underline{hs, out, \diamond}$
  if $err$ ret $\underline{\diamond, \diamond, err}$
```

Figure 7: Simple DL interface $\mathcal{N}$ for Noise. Let $h, b \geq 0$ be integers and $\mathbb{G} = \langle G \rangle$ be a represented, additive group of order n. Let $cl : \{0,1\}^b \to \mathbb{Z}_n$ and $vr : \{0,1\}^* \to \{0,1\}^*$ be functions. Procedures Write and Read are defined in Figure 8.

---

correct shared secret. This works because each key derivation depends on all shared secrets computed in the protocol so far.

The NX pattern is similar except that the public key is *transmitted* to the initiator during the handshake, rather than out-of-band. For our purposes, the significant difference between NK and NX is that, in the former pattern, the initiator confirms knowledge of the shared secret *before* the responder consumes the message and produces its response. On the other hand, in the NX pattern the initiator can send an arbitrary element of the DH group as its ephemeral key and observe a valid response without demonstrating knowledge of its discrete logarithm. This leads to information leakage beyond what is learned by honest initiators (that is, for computationally bounded attackers). It is akin to providing the adversary with a functional DH oracle, which enables an attack against the GAP1 security of the interface; as we did in Theorem 4, one can exhibit a distinguisher that gets high advantage if the IDH problem is hard for the underlying group. (More on this attack in the next section.) To reason about this attack in our analysis, we require an abstraction for handshake patterns and the actions they induce.

Definition 10 (Patterns, actions, and tokenizers)**.** A handshake pattern is a sequence of *message patterns* that specify the sequence of *tokens* processed when producing or consuming a message. A message pattern is a string that can be parsed by a *tokenizer*, which determines the set of valid *actions*. A tokenizer is a deterministic algorithm $\mathcal{T}(\textbf{bool}\, f, r, \textbf{str}\, pat) \mapsto \textbf{tup}\, \boldsymbol{t}, \textbf{str}\, err$. String pat is the message pattern, f indicates whether or not the host is producing a message, and r indicates whether the host is the initiator. The outputs are a tuple $\boldsymbol{t}$ comprised of the sequence of tokens to be processed and a string err indicating whether an error occurred. A valid action for $\mathcal{T}$ is a triple (f, r, pat) for which $err = \diamond$, where $(\boldsymbol{t}, err) = \mathcal{T}_f(r, pat)$. We say that $\mathcal{T}$ has *action count* ℓ if $|\boldsymbol{t}| \leq \ell$ for every valid action (f, r, pat).

A *token action* is a triple $(f, r, t) \in \{0,1\} \times \{0,1\} \times \{0,1\}^*$. We say that a tokenizer $\mathcal{T}$ *includes* a set of token actions $\mathcal{X}$ if for each $(f, r, t) \in \mathcal{X}$ the following is true: there exists a valid pattern pat for $\mathcal{T}$ such that $t = \boldsymbol{t}_i$ for some $1 \leq i \leq |\boldsymbol{t}|$ and $(\boldsymbol{t}, err) = \mathcal{T}_f(r, t)$. If this condition holds for no such token action, then $\mathcal{T}$ *excludes* $\mathcal{X}$. ◆

Remark 4. Our notion of message-pattern validity (relative to a tokenizer) is not rich enough to account for all handshake-pattern validity rules in the Noise specification [52, Section 7.3]. In particular, while we can say what it means for an action to be valid, our abstraction cannot account for the (in)validity of a *sequence* of actions. For example, the spec only permits the use of one ephemeral key and one static key per handshake, but this rule cannot be enforced by $\mathcal{T}$. Nevertheless, this definition is sufficient for what we aim to prove in the current work.

6.2 The interface

The interface is specified as the composition of a tokenizer and the DH, AEAD, and hash primitives. Fix integers $k, n', h, b \geq 0$ and let $\mathbb{G} = \langle G \rangle$ be a represented, additive group of order n. (The core spec mentions Curve25519 and Curve448 [52, Section 12], but other groups are permitted and used in practice.) Let $\mathcal{AE}$ be an AEAD scheme (either AES-GCM or ChaCha20-Poly1305) with key-length k and nonce-length n'. Let $cl : \{0,1\}^b \to \mathbb{Z}_n$, $vr : \{0,1\}^* \to \{0,1\}^*$, and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^h$ be functions. Function $\mathcal{H}$ is a hash function (either SHA2 or BLAKE2) that will serve multiple purposes, one of which is to derive symmetric keys using HKDF [38]. We will ignore the details of HKDF in this section and simply denote key derivation by a function $\mathcal{F} : (\{0,1\}^*)^3 \to (\{0,1\}^h)^3$ that maps an "information" string id, a "salt" X, and input key material Y to a triple of h-bit strings $\mathcal{F}(id, X, Y)$. We will model $\mathcal{F}$ as a random oracle in our analysis; in Appendix B we prove that $\mathcal{F}$ is indifferentiable [47] from an RO when $\mathcal{H}$ is modeled as an RO. We warn that this result imposes some mild restrictions on the hash function $\mathcal{H}$, which we enumerate in Appendix B. (These conditions are satisfied by the algorithms specified for use in Noise.)

The high-level specification. Figure 7 specifies our Noise interface $\mathcal{N}$ at a high level and defines structures **st** and **msg** for the handshake state and messages respectively. The key generator $\mathcal{N}$.Gen chooses a random, b-bit string K, sets $s \leftarrow cl(K)$, and returns $(\underline{sG}, \underline{s})$. (Thus, $\mathcal{N}$ is simple in the sense of Def. 8.) Function cl serves the same purpose as cl in our specification of EdDSA; it maps a bit string of a particular length to a suitable scalar s for use with the given group. (This captures the behavior of the X25519 and X448 DH functions [43] as they are used in Noise.) The key operator $\mathcal{N}$.Op is defined in terms of two procedures:

- Read(&**st** hs, **int** s, **bool** r, **str** pat, **msg** req) $\mapsto$ **str** out, err. Called when consuming an inbound message. It takes as input the static key s and processes the action $(0, r, pat)$ on the message req and current handshake state hs. It outputs a payload out.
- Write(&**st** hs, **int** s, **bool** r, **str** pat, in) $\mapsto$ **msg** $resp$, **str** err. Called when producing an outbound message. It takes as input the static key s and processes the action $(1, r, pat)$ on the payload in and current handshake state hs. It outputs a message $resp$.

Read and Write are defined (Figure 8) in terms of $\mathcal{T}$, $\mathcal{AE}$, $\mathcal{F}$, and $\mathcal{H}$. The operand encodes the current handshake state hs and the input in, and the operator op encodes an action (f, r, pat). If $f = 1$, then the host interprets in as a payload to send to its peer in its next handshake message; it calls Write and returns the updated state and outbound message. If $f = 0$, then the host interprets in as a message sent by the peer; it calls Read and returns the updated state and inbound payload.

Context-to-action binding. The context ctx is bound to the handshake state via a field $hs.id$, which should be equal to $vr(ctx)$ (7:12). We will call this the *handshake identifier.* Each call to $\mathcal{F}$ made by either Read or Write uses $hs.id$ as the label. In this way, interface $\mathcal{N}$ binds the string $hs.id = vr(ctx)$ to each key derivation, thereby binding the context to the *action* being performed. We call this *context-to-action* binding. This differs from Noise as it is, which uses an empty string as the information string for key derivation via HKDF (see [52, Section 4.3]). (Formally, the processing rules as they are specified are recovered by defining $vr(ctx) = \varepsilon$ for all ctx.) Noise binds the context to *initialization* of the handshake state (see [52, Section 5.3]), but action binding is required in our attack model in order to provide context separation when the game leaks its internal handshake state to the adversary. We will discuss the issue that arises in Section 6.3.

The low-level details. Procedures Read and Write are defined in Figure 8. Both begin by computing the sequence of tokens $\boldsymbol{t}$ from the action (f, r, pat) via $\mathcal{T}$. The tokens are then processed in order, updating the handshake state as a side-effect. Once all of the tokens have been processed, procedure Write "encapsulates" the outbound payload and attaches it to the outbound message; procedure Read "decapsulates" the inbound payload attached to the inbound message and returns it. (We will define "capsulation" in a moment.)

Available tokens are e, s, ee, es, se, ss, and psk. The token-processing rules are specified by procedures rTok and wTok (invoked by Read and Write respectively). Messages have three components (7:5): the payload blob, the ephemeral key blob, and the static key blob. On token e, the reader interprets the ephemeral key blob as the peer's ephemeral public key $hs.R$ (8:9), and the writer generates an ephemeral secret key $hs.e$ and uses $\underline{(hs.e)G}$ as its ephemeral key blob (8:37). On s, the reader decapsulates the static key blob in the inbound message (8:11), and the writer encapsulates its static public key sG and adds the blob to the outbound message. The remaining tokens update the symmetric key state $(hs.L)$ and key $(hs.K)$ and reset

```
$\mathrm{Read}(\&\mathbf{st}\ hs, \mathbf{int}\ s, \mathbf{bool}\ r, \mathbf{str}\ pat, \mathbf{msg}\ in)$
1  $\mathbf{dec\ str}\ out$; $\mathbf{int}\ i$
2  $(\boldsymbol{t}, err) \leftarrow \mathcal{T}_0(r, pat)$
3  $\mathbf{while}\ \neg err \wedge i < |\boldsymbol{t}|\ \mathbf{do}\ i \leftarrow i + 1$
4      $err \leftarrow \mathrm{rTok}(\&hs, in, s, r, \boldsymbol{t}_i)$
5  $\mathbf{if}\ err\ \mathbf{then\ ret}\ (out, err)$
6  $(out, err) \leftarrow \mathrm{Cap}_0(\&hs, in.P)$
7  $\mathbf{ret}\ (out, err)$

$\mathrm{rTok}(\&\mathbf{st}\ hs, \mathbf{msg}\ in, \mathbf{int}\ s, \mathbf{bool}\ r, \mathbf{str}\ t)$
8  $\mathbf{switch}\ (t)$
9      $\mathbf{case}$ e: $\underline{hs.R} \leftarrow in.E$
10         $\mathbf{ret}\ \mathrm{mHash}(\&hs, in.E)$
11     $\mathbf{case}$ s:
12         $(X, err) \leftarrow \mathrm{Cap}_0(\&hs, in.S)$
13         $\underline{hs.Q} \leftarrow X$
14         $\mathbf{ret}\ err$
15 $\mathbf{ret}\ \mathrm{mKeyTok}(\&hs, s, r, t)$

$\mathrm{Cap}(\mathbf{bool}\ f, \&\mathbf{st}\ hs, \mathbf{str}\ X)$
16 $\mathbf{if}\ \neg hs.K\ \mathbf{then\ ret}\ (X, \mathrm{mHash}(\&hs, X))$
17 $(K, A) \leftarrow (hs.K, hs.A)$; $N \leftarrow \underline{hs.seq}_{n'}$
18 $\mathbf{if}\ f\ \mathbf{then}\ Y \leftarrow \mathcal{AE}.\mathrm{Enc}(K, N, A, X)$
19 $\mathbf{else}\ Y \leftarrow \mathcal{AE}.\mathrm{Dec}(K, N, A, X)$
20 $\mathbf{if}\ Y = \bot\ \mathbf{then\ ret}\ (\diamond,$ err_cap$)$
21 $\mathrm{mHash}(\&hs, Y)$; $\mathbf{ret}\ (Y, \mathrm{iNonce}(\&hs))$

$\mathrm{iNonce}(\&\mathbf{st}\ hs)$
22 $\mathbf{if}\ hs.seq \geq 2^{64} - 1\ \mathbf{then}$
23     $\mathbf{ret}$ err_nonce
24 $hs.seq \leftarrow hs.seq + 1$; $\mathbf{ret}\ \diamond$

$\mathrm{mKeyPSK}(\&\mathbf{st}\ hs, \mathbf{str}\ psk)$
25 $\mathbf{if}\ \neg hs.psk\ \mathbf{then\ ret}$ err_psk
26 $(hs.L, L', L'') \leftarrow \mathcal{F}(hs.id, hs.L, psk)$
27 $hs.K \leftarrow L''[{:}k]$; $hs.seq \leftarrow 0$
28 $\mathbf{ret}\ \mathrm{mHash}(\&hs, L')$

$\mathrm{Write}(\&\mathbf{st}\ hs, \mathbf{int}\ s, \mathbf{bool}\ r, \mathbf{str}\ pat, in)$
29 $\mathbf{dec\ msg}\ out$; $\mathbf{int}\ i$
30 $(\boldsymbol{t}, err) \leftarrow \mathcal{T}_1(r, pat)$
31 $\mathbf{while}\ \neg err \wedge i < |\boldsymbol{t}|\ \mathbf{do}\ i \leftarrow i + 1$
32     $err \twoheadleftarrow \mathrm{wTok}(\&hs, \&out, s, r, \boldsymbol{t}_i)$
33 $\mathbf{if}\ err\ \mathbf{then\ ret}\ (out, err)$
34 $(out.P, err) \leftarrow \mathrm{Cap}_1(\&hs, in)$
35 $\mathbf{ret}\ (out, err)$

$\mathrm{wTok}(\&\mathbf{st}\ hs, \&\mathbf{msg}\ out, \mathbf{int}\ s, \mathbf{bool}\ r, \mathbf{str}\ t)$
36 $\mathbf{switch}\ (t)$
37     $\mathbf{case}$ e: $(out.E, ek) \twoheadleftarrow \mathrm{Gen}(\,)$
38         $hs.e \leftarrow \mathrm{Scal}(ek)$
39         $\mathbf{ret}\ \mathrm{mHash}(\&hs, out.E)$
40     $\mathbf{case}$ s:
41         $(out.S, err) \leftarrow \mathrm{Cap}_1(\&hs, \underline{sG})$
42         $\mathbf{ret}\ err$
43 $\mathbf{ret}\ \mathrm{mKeyTok}(\&hs, s, r, t)$

$\mathrm{mKeyTok}(\&\mathbf{st}\ hs, \mathbf{int}\ s, \mathbf{bool}\ r, \mathbf{str}\ t)$
44 $Y_0 \leftarrow hs.Q$; $Y_1 \leftarrow hs.R$
45 $x_0 \leftarrow s$; $x_1 \leftarrow hs.e$
46 $\mathbf{switch}\ (t)$
47     $\mathbf{case}$ psk: $\mathbf{ret}\ \mathrm{mKeyPSK}(\&hs, hs.psk)$
48     $\mathbf{case}$ ee:  $\mathbf{ret}\ \mathrm{mKeyDH}(\&hs, x_1, Y_1)$
49     $\mathbf{case}$ es:  $\mathbf{ret}\ \mathrm{mKeyDH}(\&hs, x_r, Y_{1-r})$
50     $\mathbf{case}$ se:  $\mathbf{ret}\ \mathrm{mKeyDH}(\&hs, x_{1-r}, Y_r)$
51     $\mathbf{case}$ ss:  $\mathbf{ret}\ \mathrm{mKeyDH}(\&hs, x_0, Y_0)$
52 $\mathbf{ret}$ err_token

$\mathrm{mKeyDH}(\&\mathbf{st}\ hs, \mathbf{int}\ x, \mathbf{elem}_{\mathbb{G}}\ Y)$
53 $\mathbf{if}\ \neg x \vee \neg Y\ \mathbf{then\ ret}$ err_dh
54 $(hs.L, L', L'') \leftarrow \mathcal{F}(hs.id, hs.L, \underline{xY})$
55 $hs.K \leftarrow L'[{:}k]$; $hs.seq \leftarrow 0$; $\mathbf{ret}\ \diamond$

$\mathrm{mHash}(\&\mathbf{st}\ hs, \mathbf{str}\ X)$
56 $hs.A \leftarrow \mathcal{H}(hs.A \,\|\, X)$; $\mathbf{ret}\ \diamond$
```

Figure 8: Low-level procedures for specifying $\mathcal{N}$ (Figure 7). Let $k, n', h \geq 0$ be integers such that $h \geq k$. Let $\mathcal{T}$ be a tokenizer, let $\mathcal{AE}$ be an AEAD scheme with key-length k and nonce-length n', and let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^h$ and $\mathcal{F} : (\{0,1\}^*)^3 \to (\{0,1\}^h)^3$ be functions.

---

the nonce ($hs.N$) as specified by mKeyTok. All key-update tokens but psk perform a DH operation; psk mixes a pre-shared symmetric key ($hs.psk$) into the state. All tokens but the DH operations update the shared hash state ($hs.A$) via mHash, which is used as associated data when encrypting or decrypting. Encryption is opportunistic; if Cap is called and $hs.K$ is set, then the input is encrypted if $f = 1$ and decrypted otherwise; if no key is set, then the input is passed through in plaintext. The nonce is incremented after encrypting or decrypting (8:21). When encapsulating (resp. decapsulating), the output (resp. input) is mixed into the hash state via mHash. We refer to this process as capsulation.

A GAP1 attack against any NX-capable interface. Since the static secret is only ever used for DH, we could have simply exposed mKeyDH (8:53–55) directly and left the rest of the processing logic to the application (i.e., the game). Specifying as much processing logic as we have reflects a need to carefully "wrap" this DH operator so that the interface does not give rise to a static DH oracle in our attack model.

Recall the NK pattern discussed in Section 6.1. The responder implicitly authenticates itself to the initiator by transmitting a ciphertext encrypted under the correct symmetric key; in our security analysis, we will need to exhibit an algorithm that simulates an interface performing this action without knowledge of the static secret. We will do so by modeling $\mathcal{F}$ as a random oracle and have the simulator reconstruct the correct key from the adversary's RO queries. For this argument to work, it is crucial that the initiator (i.e., the adversary) "proves" knowledge of the shared secret by, for example, encrypting the payload and attaching it to its message.

NX is an example of a pattern in which the initiator need not prove knowledge of the shared secret. A GAP1 attacker against $\mathcal{N}$ can learn $\mathcal{F}(vr(ctx), u, \underline{sR})$ for any $R \in \mathbb{G}$ and ctx it wishes and u for which it does not control, but can compute from the interface's response. Consider the following GAP1 distinguisher against an NX-capable interface. (It is similar to the wGAP2 distinguisher in Theorem 4.) The distinguisher first generates an ephemeral key pair $(\underline{E}, \underline{e}) \twoheadleftarrow \mathrm{Gen}(\,)$ and sends a message with the public key to its **Op** oracle just as it would send the first message to the responder in the NX protocol. Let $\underline{hs, resp, err}$ denote the reply. Given e, the public key P, and the contents of the response $resp$, the distinguisher can easily determine if $resp.P$ is a ciphertext encrypted under the correct symmetric key. It does so as follows. Let hs denote its handshake state prior to consuming the message. Run $err \leftarrow \mathrm{rTok}(\&hs, resp, \diamond, 1, \boldsymbol{t})$ for each t in $(\mathsf{e}, \mathsf{ee}, \mathsf{s}, \mathsf{es})$. If $err \neq \diamond$ at any point, then guess 0; otherwise guess 1. If the oracle's response is well-formed, then it is equivalent to running $hs.A \leftarrow \mathcal{H}(hs.A \,\|\, resp.E)$; $\underline{R} \leftarrow resp.E$; $(hs.L, L', *) \leftarrow \mathcal{F}(id, hs.L, \underline{eR})$; $hs.A \leftarrow \mathcal{H}(hs.A \,\|\, resp.S)$; $\underline{Q} \leftarrow \mathcal{AE}.\mathrm{Dec}(L'[k{:}], 0^n, hs.A, resp.S)$; $(hs.L, L', *) \leftarrow \mathcal{F}(id, hs.L, \underline{eQ})$; and $M \leftarrow \mathcal{AE}.\mathrm{Dec}(L'[k{:}], 0^n hs.A, resp.P)$.

We conjecture that, when HKDF is modeled as a random oracle and $\mathcal{AE}$ is INT-CTXT secure, the only way for the simulator to thwart this distinguisher is to solve the IDH problem for $\mathbb{G}$ relative to our NX-capable interface. (We will not prove it, as the details are closely related to Theorem 4.) The "unsafe action" (with respect to GAP1) is handing the interface a DH public key without proving knowledge of the corresponding secret. Related attacks are possible against any interface that induces this action. In the next section, we will prove that it suffices to exclude a small number of token actions.

Other design notes. Our interface assumes a fixed choice of parameters, and so it is not possible to negotiate these with our interface. As a result, our analysis does not address the issue of cryptographic agility [2], particularly as it pertains to the set of AEAD schemes that can be negotiated. We remark that the processing rules prevent the use of the same key with two different AEAD schemes by binding the name of the scheme to the initial state, but our analysis does not address this mechanism. Not addressing agility was a conscious choice made to focus the exposition, but which future work must address.

The operator $\mathcal{N}.\mathrm{Op}$ requires that $|hs.L| = h$ (see line 7:11). This is due to a requirement imposed on HKDF in order to prove it indifferentiable from a random oracle. We refer the reader to Appendix B for details.

Lastly, our interface only encompasses a subset of the validity rules for Noise. We only include those rules that are essential to the security goals in the current work, which we emphasize are orthogonal to the security goals of particular protocols (i.e., (mutual) entity authentication or authenticated key exchange, forward secrecy, etc.). Validity rules that are needed to achieve these goals would be enforced by the game.

6.3 Security

Interface $\mathcal{N}$ is GAP1 secure for any game $\mathcal{G}$ subject to the following restrictions. First, the tokenizer must exclude any write action involving DH on the static secret. (It may, however, read messages that depend on the static secret.) And second, each time $\mathcal{G}$ invokes $\mathcal{F}$ on an input (id, u, v) it must hold that $id = vr(\alpha)$, where α is the game context.

Fix Noise parameters $(\mathbb{G}, \mathcal{AE}, \mathcal{T}, \mathcal{H}, \mathcal{F}, cl, vr, k, n', h, b)$ and let $\mathcal{N}$ be the DL interface instantiated with these parameters as specified in Figure 7. Let $n = |\mathbb{G}|$ and let $\mathcal{X} = \{(1, 0, \mathsf{es}), (1, 0, \mathsf{ss}), (1, 1, \mathsf{se}), (1, 1, \mathsf{ss})\}$. Define $\psi : \{0,1\}^* \times (\{0,1\}^*)^3 \to \{0,1\}$ as the map $(ctx, (id, u, v)) \mapsto (vr(ctx) = id)$.

Theorem 7. *Suppose that n is prime. Let $\mathcal{G}$ be a regular game and suppose that $\mathcal{T}$ is $\mathcal{X}$-excluding and has action count ℓ. Let* **DDH** *be as defined in Figure 4. When $\mathcal{F}$ is modeled as a random oracle, there exists a regular,* **DDH**- *and* **Q**-*relative simulator $\mathcal{S}$ such that for all $t, q_G, q_I, q_R, c \geq 0$ there exists a $\hat{t}$-time CR-adversary $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\mathrm{gap1}}_{\mathcal{N},\mathcal{G}}(\mathcal{S}, \boldsymbol{r}) \leq 2cq_I\mathbf{Adv}^{\mathrm{cr}}_{vr}(\mathcal{C}) + 2\ell q_I \mathbf{Adv}^{\text{int-ctxt}}_{\mathcal{AE}}(\hat{t}, 0, q_I)\,,$$

*where $\mathcal{G}$ is c-ro-bound and ψ-ro-regular; $\mathcal{AE}$, $\mathcal{T}$, $\mathcal{H}$, cl, and vr are 0-ro-bound; simulator $\mathcal{S}$ is $(O(t/(q_I + 1)), q_I, \ell)$-resource, $\ell q_R q_I$-***DDH***-bound, and 2-***Q***-bound; $\boldsymbol{r} = (t, q_G, q_I, q_R)$; and $\hat{t} = O(t + q_R q_I)$.*

We will sketch the main ideas of the proof; refer to Appendix A.6 for the details. To simulate static DH computations on an input Y (either the peer's static or ephemeral key), the simulator $\mathcal{S}$ computes the set $\mathcal{V}$ of points incident to the adversary's RO queries. For each $Z \in \mathcal{V}$ it uses its DDH oracle to check if $(\log_G P)(\log_G Y) = \log_G Z$, where P is the host's static key. If so, then it uses Z to simulate the output of the interface. This is only possible in general for read actions, since these require the adversary to compute a ciphertext under the correct symmetric key, which can be obtained by querying the RO first. In fact, what we show is that, short of breaking the CR security of vr or INT-CTXT security of $\mathcal{AE}$, the only way to get a valid response from **Op** is to compute the inbound message as specified by the processing rules.

The need for context-to-action binding and the restriction of the game's RO queries arise in order to ensure there is no "subliminal channel" between the game and the adversary conveying information about the RO to the adversary beyond what it learns by making RO queries on its own. If the game provides the outputs of its RO queries to the adversary (e.g., by compromising the handshake state), then without action binding, these can be used by the adversary to compute ciphertexts without interacting with the RO. Hence, there is no way for the simulator to correctly respond given only knowledge of the adversary's RO queries. (Allowing the simulator to observe more RO queries than this—in particular, the game's—would make composition impossible.)

Next, as we did in Section 5.2, we apply the GAP1 security of $\mathcal{N}$ and the composition theorem to the IDH problem for $\mathcal{N}$. We cannot reduce the CDH problem to it as we did in Corollary 1, since the simulator requires a **DDH** oracle. Of course, this is precisely what the GDH experiment provides. The following is obtained by applying Theorems 1 and 7. (We will not prove it, but the details are closely related to Corollary 1.)

Corollary 2. *Suppose that n is prime and that $\mathcal{T}$ is $\mathcal{X}$-excluding and has maximum action count ℓ. Let $r = |\mathrm{Rng}\, cl|$ and suppose that $r \mid 2^b$. Then for all $t, q_I, q_R \ge 0$ it holds that*

$$\mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{N}}(t, q_I, q_R) \le n/r\,\mathbf{Adv}^{\mathrm{gdh}}_{\mathbb{G}}(O(t+\hat{q}), \ell q_R q_I) + 2\ell q_I\,\mathbf{Adv}^{\mathrm{int\text{-}ctxt}}_{\mathcal{AE}}(\hat{t}, 0, q_I)\,,$$

where $\mathcal{F}$ is modeled as a random oracle; $\mathcal{AE}$, $\mathcal{T}$, $\mathcal{H}$, cl, and vr are 0-ro-bound; $\hat{q} = q_R + \ell(q_I + 1)$; and $\hat{t} = O(t + q_R q_I)$.

Remark 5. The use of the **DDH** oracle by the simulator in Theorem 7 is standard; it is used, for instance, to prove joint security of encryption and signing in the ROM [22]. In fact, the Noise spec calls for a group for which the GDH problem is hard; see [52, Section 4.1]. However, we are not certain that the DDH oracle is essential to the argument. The current proof uses the DDH oracle and the adversary's RO queries to determine if the adversary knows the symmetric key used to encrypt the payload currently being processed. It may be possible to simply check each symmetric key output by the adversary's RO queries if the key correctly decrypts the ciphertext. However, this argument would require that $\mathcal{AE}$ be "key-robust" in the sense of Farshim, Orlandi, and Rosie [28], which demands, roughly, that it is difficult to find (K_1, K_2, N, A, C) such that $K_1 \ne K_2$ and (N, A, C) decrypts properly under both K_1 and K_2. AEAD schemes are not usually designed to have this property; in particular, AES-GCM is not key robust due to an attack by Dodis et al. [23].

6.4 Composition with EdDSA

In Section 5.2 we proved (modeling the underlying hash function as a random oracle) that EdDSA can be securely composed with any simple DL interface. Because $\mathcal{N}$ is simple, by Theorem 1(ii) if $\mathcal{N}$ is SEC/I secure for a game $\mathcal{G}$, then so is their composition, which we denote $\mathcal{ED}_{+\mathcal{N}}$. (We write it this way to denote the fact that it uses $\mathcal{ED}$'s key generator rather than $\mathcal{N}$'s.) We would also like to prove the complimentary statement: that if $\mathcal{ED}$ is secure for a game $\mathcal{G}$, then so is $\mathcal{ED}_{+\mathcal{N}}$.

This follows from a proof of GAP2 security of $(\mathcal{I}_{+\mathcal{N}}, \mathcal{I})$, where $\mathcal{I}_{+\mathcal{N}} = (\mathcal{I}.\mathrm{Op}, \mathrm{Op})$ is an interface constructed from $\mathcal{N}$ and DL interface $\mathcal{I}$ as follows. On input of (sk, ctx, op, in), the operator interprets op as $\underline{u, f, r, pat}$, where u is a string and (f, r, pat) is an action. If $u = \mathsf{noise}$, then return $\mathcal{N}.\mathrm{Op}(\underline{s}, ctx, op, in)$, where $s = \mathcal{I}.\mathrm{Scal}(sk)$; otherwise return $\mathcal{I}.\mathrm{Op}(sk, ctx, op, in)$. We can prove this using essentially the same argument as in Theorem 7, but we also need to account for $\mathcal{I}$'s RO queries. Fix Noise parameters $(\mathbb{G}, \mathcal{AE}, \mathcal{T}, \mathcal{H}, \mathcal{F}, cl, vr, k, n', h, b)$ and fix token-action set $\mathcal{X}$ and predicate ψ as in the previous section.

Theorem 8. *Suppose that* n *is prime. Let* $\mathcal{G}$ *be a regular game and suppose that* $\mathcal{T}$ *is* $\mathcal{X}$*-excluding and has action count* ℓ*. Let* $\mathbf{DDH}$ *be as defined in Figure 4. When* $\mathcal{F}$ *is modeled as a random oracle, there exists a* $\mathbf{DDH}$*- and* $\mathbf{Q}$*-relative regular simulator* $\mathcal{S}$ *such that for all* $t, q_G, q_I, q_R, c \geq 0$ *there exists a* $\hat{t}$*-time CR-adversary* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{gap2}}_{\mathcal{I}_{+\mathcal{N}},\mathcal{I},\mathcal{G}}(\mathcal{S}, \boldsymbol{r}) \leq 2cq_I \mathbf{Adv}^{\text{cr}}_{vr}(\mathcal{C}) + 2\ell q_I \mathbf{Adv}^{\text{int-ctxt}}_{\mathcal{AE}}(\hat{t}, 0, q_I)\,,$$

where $\mathcal{G}$ *is* c*-ro-bound and* ψ*-ro-regular;* $\mathcal{I}$ *is* ψ*-ro-regular and* $\mathcal{I}.\mathrm{Gen}$ *is* 0*-ro-bound;* $\mathcal{AE}$*,* $\mathcal{T}$*,* $\mathcal{H}$*,* cl*, and* vr *are* 0*-ro-bound; simulator* $\mathcal{S}$ *is* $(O(t/(q_I+1)), q_I, \ell)$*-resource,* $\ell q_R q_I$*-*$\mathbf{DDH}$*-bound, and* 2*-*$\mathbf{Q}$*-bound;* $\boldsymbol{r} = (t, q_G, q_I, q_R)$*; and* $\hat{t} = O(t + q_R q_I)$*.*

(Note that the vr parameter in the CR-advantage term is the one for Noise and not for EdDSA.) We sketch the proof in Appendix A.7. By Theorem 1 we conclude that if $\mathcal{ED}$ is SEC/I secure for a game $\mathcal{G}$ that meets the conditions of Theorem 8, then so is $\mathcal{ED}_{+\mathcal{N}}$.

Remark 6. In Theorem 6 we modeled the underlying hash function as a random oracle, and in Theorem 8 we modeled the HKDF as a random oracle. Indeed, since HKDF may be constructed from the very same hash function used with EdDSA, it is crucial that HKDF is indifferentiable from an RO when the hash function is modeled as an RO. Otherwise, there may be an attack against $\mathcal{ED}_{+\mathcal{N}}$ with respect to some game $\mathcal{G}$ that exploits the underlying structure of HKDF; such an attack would be out-of-scope of our analysis. Fortunately, we are able to rule these out in Appendix B.

Discussion. So far, our treatment has elided an important detail regarding the composition of EdDSA and Noise. Both are specified in terms of a generic group, and so for their composition to make sense, this must be the same group for both interfaces. The curves used by Noise, Curve25519 and Cure448, are *birationally equivalent* to the Edwarsd curves used by Ed25519 and Ed448 respectively [14]. Loosely speaking, this means there is an efficiently computable map from points on one curve to points on the other. This map would be used to transform the EdDSA key pair into a static key pair for Noise. We emphasize, however, that due to our formalization of represented groups, the use of this transform is out-of-scope of our analysis.

6.5 Conclusion

Theorem 7 specifies conditions for $\mathcal{N}$ and $\mathcal{G}$ that suffice for $\mathcal{N}$ to be SEC/I secure for $\mathcal{G}$. The restriction to (ψ-ro-)regular games $\mathcal{G}$ is mild, but proving GAP1 security—at least, without making additional restrictions on the protocol and security goal captured by $\mathcal{G}$—requires modifying the processing rules. But the change we suggest is a relatively simple one. Noise already provides a degree of context separability by binding the handshake pattern and parameters to the initial state (see [52, Sec. 5.2]). This is provided by using the hash of the protocol name (e.g. Noise_NK_25519_AESGCM_SHA256) as the initial HKDF salt. Our suggestion is that by simply using the protocol name as the HKDF label instead, Noise protocols would be context separable under a much wider set of circumstances (in particular, all conditions captured by $\mathcal{G}$ in Theorem 7). The more consequential finding of our analysis is that, in order to prove GAP1 security, we must exclude handshake patterns that entail certain actions that are insecure with respect to GAP1 (in particular, the set of token actions $\mathcal{X}$). By our count, of the 37 one-way, fundamental, and deferred handshake patterns in the spec [52], 19 would be supported by the initiator, 24 by the responder, and only 10 by both. We stress, however, that we are not aware of an explicit SEC/I attack at this time. It is likely that, in many cases, our analysis is more conservative than is necessary.

As a result of these restrictions, our work leaves the security of key-reuse among Noise protocols *as they are* an open question. In particular, our work leaves open the possibility of a direct proof of SEC/I security with respect to a *particular* game. One could also achieve meaningful results by considering joint security in a weaker attack model wherein parties may negotiate one of a fixed set of concrete Noise patterns. However, these results would be far less general than ours.

The primary goal of this work was not to analyze Noise as it is, but to demonstrate that the analytical approach we developed can be applied to reason about protocol interactions in a very general way. The design philosophy underlying Noise (build the framework first, then build protocols from the framework) makes Noise an ideal case study. (Indeed, a similar analysis could be carried out in order to shed light on protocol interactions among other DH-based protocols [32, 40, 42, 44].) It is not surprising that our analysis would unearth limitations with respect to our security goals, since, to the best of our knowledge, these goals have not been considered before.

(In)security with respect to GAP1 is akin to (in)security in the sense of indifferentiability [47]. For example, while length-extension attacks against Merkle-Damgård-style hash functions rule out indifferentiability from a random oracle [21], whether these attacks are exploitable depends on how the hash function is used. Analogously, whether the vulnerabilities of Noise with respect to GAP1 are exploitable in practice depends intrinsically upon the details of the protocol implemented by the interface, as well as the intended security goal. On the other hand, and analogous to indifferentiability, security in our setting rules out these vulnerabilities altogether.

Finally, we believe the design philosophy underlying Noise has the potential to re-shape how our community approaches protocol design and analysis. It is our hope that this work will help lay the formal foundations for this effort going forward.

Acknowledgements

This work was made possible by NSF grant CNS-1816375. We thank the anonymous reviewers for their comments. We thank Trevor Perrin for his valuable feedback on our analysis of Noise. We also thank Stefano Tessaro for his input on the indifferentiability of HKDF (Appendix B). Thanks to Dave Tian for providing his systems security perspective on the problem, and thanks to Luis Vargas and Tyler Tucker for editorial feedback.

References

[1] Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Topics in Cryptology — CT-RSA 2001. pp. 143–158. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)

[2] Acar, T., Belenkiy, M., Bellare, M., Cash, D.: Cryptographic agility and its relation to circular encryption. In: Advances in Cryptology – EUROCRYPT 2010. pp. 403–422. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

[3] Acar, T., Nguyen, L., Zaverucha, G.: A TPM Diffie-Hellman oracle. Cryptology ePrint Archive, Report 2013/667 (2013), `https://eprint.iacr.org/2013/667`

[4] Acar, Y., Backes, M., Fahl, S., Garfinkel, S., Kim, D., Mazurek, M.L., Stransky, C.: Comparing the usability of cryptographic APIs. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 154–171 (May 2017)

[5] Adams, A., Sasse, M.A.: Users are not the enemy. Commun. ACM **42**(12), 40–46 (Dec 1999)

[6] Albrecht, M., Farshim, P., Paterson, K., Watson, G.: On cipher-dependent related-key attacks in the ideal-cipher model. In: Fast Software Encryption 2011 – FSE 2011. pp. 128–145. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[7] Barnes, R., Iyengar, S., Sullivan, N., Rescorla, E.: Delegated credentials for TLS. Internet-Draft draft-ietf-tls-subcerts-03, IETF Secretariat (February 2019), `http://www.ietf.org/internet-drafts/draft-ietf-tls-subcerts-03.txt`

[8] Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Advances in Cryptology - CRYPTO 2007. pp. 535–552. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

[9] Bellare, M., Brakerski, Z., Naor, M., Ristenpart, T., Segev, G., Shacham, H., Yilek, S.: Hedged public-key encryption: How to protect against bad randomness. In: Advances in Cryptology – ASIACRYPT 2009. pp. 232–249. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[10] Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In: Advances in Cryptology – EUROCRYPT 2003. pp. 491–506. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)

[11] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. pp. 62–73. CCS '93, ACM, New York, NY, USA (1993)

[12] Bellare, M., Rogaway, P.: The security of triple encryption and a frameworkforcode-basedgame-playingproofs. In: Advances in Cryptology - EUROCRYPT 2006. pp. 409–426. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

[13] Bellare, M., Tackmann, B.: The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In: Advances in Cryptology – CRYPTO 2016. pp. 247–276. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)

[14] Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering **2**(2), 77–89 (Sep 2012)

[15] Bhargavan, K., Boureanu, I., Fouque, P., Onete, C., Richard, B.: Content delivery over TLS: a cryptographic analysis of Keyless SSL. In: 2017 IEEE European Symposium on Security and Privacy (EuroS P). pp. 1–16 (April 2017). https://doi.org/10.1109/EuroSP.2017.52

[16] Black, J., Rogaway, P., Shrimpton, T.: Encryption scheme security in the presense of key-dependent messages. In: International Workshop on Selected Areas in Cryptography – SAC 2002. pp. 62–75. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)

[17] Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the rsa encryption standard PKCS #1. In: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology. pp. 1–12. CRYPTO '98, Springer-Verlag, London, UK, UK (1998)

[18] Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security. pp. 132–145. CCS '04, ACM, New York, NY, USA (2004)

[19] Brown, D.R.L., Gallant, R.P.: The static Diffie-Hellman problem. Cryptology ePrint Archive, Report 2004/306 (2004), `https://eprint.iacr.org/2004/306`

[20] Camenisch, J., Chen, L., Drijvers, M., Lehmann, A., Novick, D., Urian, R.: One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 901–920 (May 2017)

[21] Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-damgård revisited: How to construct a hash function. In: Advances in Cryptology – CRYPTO 2005. pp. 430–448. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[22] Degabriele, J.P., Lehmann, A., Paterson, K.G., Smart, N.P., Strefler, M.: On the joint security of encryption and signature in EMV. In: Topics in Cryptology – CT-RSA 2012. pp. 116–135. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[23] Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: From invisible salamanders to encryptment. Cryptology ePrint Archive, Report 2019/016 (2019), `https://eprint.iacr.org/2019/016`

[24] Dodis, Y., Ristenpart, T., Steinberger, J., Tessaro, S.: To hash or not to hash again? (In)Differentiability results for H^2 and HMAC. In: Advances in Cryptology – CRYPTO 2012. pp. 348–366. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[25] Dowling, B., Paterson, K.G.: A cryptographic analysis of the wireguard protocol. In: Applied Cryptography and Network Security. pp. 3–21. Springer International Publishing, Cham (2018)

[26] Egele, M., Brumley, D., Fratantonio, Y., Kruegel, C.: An empirical study of cryptographic misuse in Android applications. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. pp. 73–84. CCS '13, ACM, New York, NY, USA (2013)

[27] Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., Smith, M.: Why Eve and Mallory love Android: An analysis of Android SSL (in)security. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 50–61. CCS '12, ACM, New York, NY, USA (2012)

[28] Farshim, P., Orlandi, C., Rosie, R.: Security of symmetric primitives under incorrect usage of keys. IACR Transactions on Symmetric Cryptology **2017**(1), 449–473 (Mar 2017). https://doi.org/10.13154/tosc.v2017.i1.449-473

[29] Fersch, M., Kiltz, E., Poettering, B.: On the provable security of (EC)DSA signatures. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1651–1662. CCS '16, ACM, New York, NY, USA (2016)

[30] Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., Shmatikov, V.: The most dangerous code in the world: Validating SSL certificates in non-browser software. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 38–49. CCS '12, ACM, New York, NY, USA (2012)

[31] Gleeson, S., Zimman, C.: PKCS #11 cryptographic token interface base specification version 2.40. Online white paper (July 2015), `http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html`

[32] Goldberg, I., Stebila, D., Ustaoglu, B.: Anonymity and one-way authentication in key exchange protocols. Designs, Codes and Cryptography **67**(2), 245–269 (May 2013). https://doi.org/10.1007/s10623-011-9604-z, `https://doi.org/10.1007/s10623-011-9604-z`

[33] Green, M., Smith, M.: Developers are not the enemy!: The need for usable security APIs. IEEE Security Privacy **14**(5), 40–46 (Sept 2016)

[34] Haber, S., Pinkas, B.: Securely combining public-key cryptosystems. In: Proceedings of the 8th ACM Conference on Computer and Communications Security. pp. 215–224. CCS '01, ACM, New York, NY, USA (2001)

[35] Josefsson, S., Liusvaara, I.: Edwards-curve digital signature algorithm (EdDSA). RFC 8032, RFC Editor (January 2017)

[36] Kelsey, J., Schneier, B., Wagner, D.: Protocol interactions and the chosen protocol attack. In: Security Protocols. pp. 91–104. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)

[37] Kobeissi, N., Bhargavan, K.: Noise Explorer: Fully automated modeling and verification for arbitrary Noise protocols. Cryptology ePrint Archive, Report 2018/766 (2018), `https://eprint.iacr.org/2018/766`

[38] Krawczyk, H., Eronen, P.: HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869, RFC Editor (May 2010), `http://www.rfc-editor.org/rfc/rfc5869.txt`

[39] Krawczyk, H., Bellare, M., Canetti, R.: Hmac: Keyed-hashing for message authentication. RFC 2104, RFC Editor (February 1997), `http://www.rfc-editor.org/rfc/rfc2104.txt`, `http://www.rfc-editor.org/rfc/rfc2104.txt`

[40] Krawczyk, H., Wee, H.: The OPTLS protocol and TLS 1.3. Cryptology ePrint Archive, Report 2015/978 (2015), `https://eprint.iacr.org/2015/978`

[41] Künnemann, R., Steel, G.: YubiSecure? formal security analysis results for the Yubikey and YubiHSM. In: Security and Trust Management. pp. 257–272. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[42] LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange (January 2007), `https://www.microsoft.com/en-us/research/publication/stronger-security-of-authenticated-key-exchange/`

[43] Langley, A., Hamburg, M., Turner, S.: Elliptic curves for security. RFC 7748, RFC Editor (January 2016)

[44] Lauter, K., Mityagin, A.: Security analysis of KEA authenticated key exchange protocol. Cryptology ePrint Archive, Report 2005/265 (2005), `https://eprint.iacr.org/2005/265`

[45] Lim, C.H., Lee, P.J.: A key recovery attack on discrete log-based schemes using a prime order subgroup. In: Advances in Cryptology — CRYPTO '97. pp. 249–263. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

[46] Lipp, B., Blanchet, B., Bhargavan, K.: A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol. Research Report RR-9269, Inria Paris (Apr 2019), `https://hal.inria.fr/hal-02100345`

[47] Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Theory of Cryptography. pp. 21–39. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

[48] Namprempre, C., Rogaway, P., Shrimpton, T.: Reconsidering generic composition. In: Advances in Cryptology – EUROCRYPT 2014. pp. 257–274. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[49] Okamoto, T., Pointcheval, D.: The gap-problems: A new class of problems for the security of cryptographic schemes. In: Public Key Cryptography. pp. 104–118. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)

[50] Oliveira, D., Rosenthal, M., Morin, N., Yeh, K.C., Cappos, J., Zhuang, Y.: It's the psychology stupid: How heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots. In: Proceedings of the 30th Annual Computer Security Applications Conference. pp. 296–305. ACSAC '14, ACM, New York, NY, USA (2014)

[51] Oliveira, D.S., Lin, T., Rahman, M.S., Akefirad, R., Ellis, D., Perez, E., Bobhate, R., DeLong, L.A., Cappos, J., Brun, Y.: API blindspots: Why experienced developers write vulnerable code. In: Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018). pp. 315–328. USENIX Association, Baltimore, MD (2018)

[52] Perrin, T.: The Noise protocol framework. Online white paper (July 2018), `https://noiseprotocol.org/noise.html`

[53] Pollard, J.M.: Kangaroos, monopoly and discrete logarithms. J. Cryptol. **13**(4), 437–447 (2000)

[54] Rescorla, E.: The transport layer security (TLS) protocol version 1.3. RFC 8446, RFC Editor (August 2018)

[55] Ristenpart, T., Shacham, H., Shrimpton, T.: Careful with composition: Limitations of the indifferentiability framework. In: Advances in Cryptology – EUROCRYPT 2011. pp. 487–506. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[56] Rogaway, P., Stegers, T.: Authentication without elision: Partially specified protocols, associated data, and cryptographic models described by code. In: 2009 22nd IEEE Computer Security Foundations Symposium. pp. 26–39 (July 2009)

[57] Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4**(3), 161–174 (Jan 1991)

[58] Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Advances in Cryptology — EUROCRYPT '97. pp. 256–266. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

[59] Shrimpton, T., Stam, M., Warinschi, B.: A modular treatment of cryptographic APIs: The symmetric-key case. In: Advances in Cryptology – CRYPTO 2016. pp. 277–307. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)

[60] Trusted Computing Group: TPM 2.0 library specification (September 2016), `https://trustedcomputinggroup.org/resource/tpm-library-specification/`

A Proofs

A.1 Theorem 1 (composition)

We will use the following lemma.

Lemma 1. *Let $t \geq 0$ be an integer, let $\mathcal{R}$ be a finite set, and let X and Y be random variables with support $\mathcal{R}$. For every t-time algorithm $\mathcal{D}(\mathbf{elem}_{\mathcal{R}}\, R) \mapsto \mathbf{bool}\, b$ it holds that $\Pr\left[\, \mathcal{D}(X) \,\right] - \Pr\left[\, \mathcal{D}(Y) \,\right] \leq \Delta(X, Y)$.*

Proof. The following is based on Daniel Wichs' lecture notes.[4] Let $\mathit{\Pi}$ denote the classs of algorithms that take as input an element of $\mathcal{R}$ and output a bit. Fix integer $t \geq 0$ and let $\mathit{\Pi}_t$ be the set of t-time algorithms $\mathcal{D}$ such that $\mathcal{D} \in \mathit{\Pi}$. There exists an integer $\rho \leq t$ such that each $\mathcal{D} \in \mathit{\Pi}_t$ consumes at most ρ random coins on any execution. Let R and S be random variables independently and uniformly distributed over $\{0,1\}^\rho$, let $B_1 = (X, R)$, and let $B_0 = (Y, S)$. Let $\mathcal{V} = \mathcal{R} \times \{0,1\}^\rho$ and let $\mathcal{W}_{\mathcal{D}} = \{(W, R) \in \mathcal{V} : \mathcal{D}(W; R) = 1\}$ and $\delta_{\mathcal{D}} = \Pr\left[\, \mathcal{D}(X) \,\right] - \Pr\left[\, \mathcal{D}(Y) \,\right]$ for every $\mathcal{D} \in \mathit{\Pi}_t$. Then

$$\begin{aligned}
\delta_{\mathcal{D}} &\leq \max_{\mathcal{D}' \in \mathit{\Pi}_t} \left(\Pr\left[\, B_1 \in \mathcal{W}_{\mathcal{D}'} \,\right] - \Pr\left[\, B_0 \in \mathcal{W}_{\mathcal{D}'} \,\right]\right) && (1)\\
&\leq \max_{\mathcal{W} \subseteq \mathcal{V}} \left(\Pr\left[\, B_1 \in \mathcal{W} \,\right] - \Pr\left[\, B_0 \in \mathcal{W} \,\right]\right) && (2)\\
&= \Pr\left[\, B_1 \in \mathcal{W}^* \,\right] - \Pr\left[\, B_0 \in \mathcal{W}^* \,\right] && (3)
\end{aligned}$$

for every such $\mathcal{D}$, where $\mathcal{W}^*$ denotes the subset of $\mathcal{V}$ that maximizes the quantity on the right hand side of Eq. (2). Note that either $\mathcal{W}^* = \mathcal{T}$ or $\mathcal{W}^* = \mathcal{V} \setminus \mathcal{T}$, where $\mathcal{T} = \{b \in \mathcal{V} : \Pr\left[\, B_1 = b \,\right] - \Pr\left[\, B_0 = b \,\right] \geq 0\}$. But $\Pr\left[\, B_1 \in \mathcal{T} \,\right] + \Pr\left[\, B_1 \in \mathcal{V} \setminus \mathcal{T} \,\right] = \Pr\left[\, B_0 \in \mathcal{T} \,\right] + \Pr\left[\, B_0 \in \mathcal{V} \setminus \mathcal{T} \,\right]$, which implies that

$$\begin{aligned}
\delta_{\mathcal{D}} &\leq \Pr\left[\, B_1 \in \mathcal{T} \,\right] - \Pr\left[\, B_0 \in \mathcal{T} \,\right] && (4)\\
&= \frac{1}{2}\left(\Pr\left[\, B_1 \in \mathcal{T} \,\right] - \Pr\left[\, B_0 \in \mathcal{T} \,\right] + \Pr\left[\, B_0 \in \mathcal{V} \setminus \mathcal{T} \,\right] - \Pr\left[\, B_1 \in \mathcal{V} \setminus \mathcal{T} \,\right]\right) && (5)\\
&= \frac{1}{2}\left(\sum_{v \in \mathcal{T}} \left(\Pr\left[\, B_1 = v \,\right] - \Pr\left[\, B_0 = v \,\right]\right) + \sum_{v \in \mathcal{V} \setminus \mathcal{T}} \left(\Pr\left[\, B_0 = v \,\right] - \Pr\left[\, B_1 = v \,\right]\right)\right) && (6)\\
&\leq \frac{1}{2} \sum_{v \in \mathcal{V}} \left|\Pr\left[\, B_1 = v \,\right] - \Pr\left[\, B_0 = v \,\right]\right| && (7)\\
&= \frac{2^\rho}{2} \sum_{v \in \mathcal{R}} \left|\Pr\left[\, X = v \,\right] \frac{1}{2^\rho} - \Pr\left[\, Y = v \,\right] \frac{1}{2^\rho}\right| && (8)\\
&= \frac{1}{2} \sum_{v \in \mathcal{R}} \left|\Pr\left[\, X = v \,\right] - \Pr\left[\, Y = v \,\right]\right| && (9)
\end{aligned}$$

for every $\mathcal{D} \in \mathit{\Pi}_t$, where Eq. (8) follows from the observation that for every $v \in \{0,1\}^\rho$, the probability that $R = v$ is $2^{-\rho}$. (Also for S.) □

We only prove condition (ii) explicitly; condition (i) follows from a similar argument. (We will discuss the differences at the end.) Let $\mathcal{A}$ be an $\boldsymbol{r}$-resource SEC/I-adversary. Consider the GAP2-adversary $\mathcal{D}$ defined by simulating $\mathcal{A}$ in the SEC/I experiment for $\mathcal{G}$ as follows. It runs $\mathcal{A}$ on its input *pk*: when $\mathcal{A}$ asks α of its **Init** oracle, $\mathcal{D}$ returns the output of **Init**(α); when $\mathcal{A}$ asks $(\mathit{ctx}, \mathit{op}, \mathit{in})$ of **Op**, $\mathcal{D}$ returns **Op**$(\mathit{ctx}, \mathit{op}, \mathit{in})$

---

[4] `http://www.ccs.neu.edu/home/wichs/class/crypto-fall15/lecture2.pdf`, accessed 2018-02-04.

```
P(X, V)                                                 $\mathbf{G}_0^j$ $\boxed{\mathbf{G}_1^j}$
1 if $\pi[X] \neq \diamond$ then $bad \leftarrow 1$; $\boxed{\textbf{ret } \bot}$
2 $i \leftarrow i + 1$; $Y_i \leftarrow V$
3 $U_i \twoheadleftarrow \{0,1\}^h$; $\pi[X] \leftarrow U_i$
4 if $i \le j$ then $\pi[X] \leftarrow Y_i$
```

```
P(X, V)                                                 $\mathbf{G}_1^j$ $\mathbf{G}_2$
5 if $\pi[X] \neq \diamond$ then $bad \leftarrow 1$; ret $\bot$
6 $i \leftarrow i + 1$; $Y_i \leftarrow V$
7 $U_i \twoheadleftarrow \{0,1\}^h$; $\pi[X] \leftarrow U_i$
8 if $i \le j$ then $\pi[X] \leftarrow Y_i$
```

Figure 9: Specification of oracle **P** in experiments 0, 1, and 2 for the proof of Theorem 1.

if *ctx* $\neq \alpha$ and $\bot$ otherwise; when $\mathcal{A}$ asks *in* of **Call**, $\mathcal{D}$ returns the output of **Call**(*in*); and when $\mathcal{A}$ asks X of **R**, $\mathcal{D}$ returns $\mathbf{R}(X)$. Finally, when $\mathcal{A}$ asks *in* of **Final**, if **Final**(*in*) outputs 1, then $\mathcal{D}$ halts and outputs 1; otherwise it halts and outputs 0. Let d_{b1} denote the probability that $\mathcal{D}$ outputs 1 given that b is the challenge bit in its experiment. Then $d_{11} = \Pr\big[\,\mathbf{Exp}^{\text{sec/i}}_{\mathcal{I}^1,\mathcal{G}}(\mathcal{A})\,\big]$, since $\mathcal{D}$ perfectly simulates $\mathcal{A}$ in the SEC/I experiment with $\mathcal{G}$ and $\mathcal{I}^1$ when $b = 1$.

Let $j \geq 0$ be an integer and let $\mathbf{G}_0^j(\mathcal{S}, \mathcal{A})$ be an experiment defined just like $\mathbf{Exp}^{\text{sec/i}}_{\mathcal{I}^0,\mathcal{G}}(\mathcal{A})$, but with the following changes: first, line 2:12 is replaced with **ret** $\langle \mathcal{S}_\sigma : \mathcal{I}^{0\mathbf{R}}_{sk}, \mathbf{P}, \mathbf{Q}, \mathbf{R}\rangle(ctx, op, in)$, where $\sigma \twoheadleftarrow \langle \mathcal{S}.\text{Init}: \mathbf{P}, \mathbf{Q}, \mathbf{R}\rangle(pk)$ was executed prior to running $\mathcal{A}$; second, the experiment declares variables **int** i and **str** $U[\,], Y[\,]$ prior to executing $\mathcal{S}$.Init; and third, the **P** oracle is as defined in Figure 9.

Note that $d_{01} = \Pr\big[\,\mathbf{G}_0^P(\mathcal{S}, \mathcal{A})\,\big]$, where $P = c_P(q_I + 1)$ is the maximum number of **P** queries made during the course of the experiment. Then

$$\begin{aligned}
\mathbf{Adv}^{\text{sec/i}}_{\mathcal{I}^1,\mathcal{G}}(\mathcal{A}) &= \Pr\big[\,\mathbf{Exp}^{\text{sec/i}}_{\mathcal{I}^1,\mathcal{G}}(\mathcal{A})\,\big] - \Pr\big[\,\mathbf{G}_0^P(\mathcal{S}, \mathcal{A})\,\big] + \Pr\big[\,\mathbf{G}_0^P(\mathcal{S}, \mathcal{A})\,\big] &\quad (10)\\
&= d_{11} - d_{01} + \Pr\big[\,\mathbf{G}_0^P(\mathcal{S}, \mathcal{A})\,\big] &\quad (11)\\
&= \mathbf{Adv}^{\text{gap2}}_{\mathcal{I}^1,\mathcal{I}^0,\mathcal{G}}(\mathcal{S}, \mathcal{D}) + \Pr\big[\,\mathbf{G}_0^P(\mathcal{S}, \mathcal{A})\,\big]\,. &\quad (12)
\end{aligned}$$

We proceed by a game-playing argument [12]. Define a new experiment $\mathbf{G}_1^j$ from $\mathbf{G}_0^j$ by modifying the **P** oracle as shown in Figure 9. The only difference between $\mathbf{G}_0^j$ and $\mathbf{G}_1^j$ is that, in the latter experiment, if **P** is queried on (X, V) such that $\pi[X]$ was previously defined, then the oracle **P** returns $\bot$ without overwriting the value of $\pi[X]$. By the Fundamental Lemma of Game Playing [12] it follows that

$$\Pr\big[\,\mathbf{G}_0^P(\mathcal{S}, \mathcal{A})\,\big] - \Pr\big[\,\mathbf{G}_1^P(\mathcal{S}, \mathcal{A})\,\big] \le \Pr\big[\,\mathbf{G}_1^P(\mathcal{S}, \mathcal{A})\,\textbf{sets}\ bad\,\big] \le q_R P / 2^{\mu_1} \qquad (13)$$

since q_R is the maximum number of **R** queries over the course of the experiment. Next, note that we have $\Pr\big[\,\mathbf{G}_2(\mathcal{S}, \mathcal{D})\,\big] = \Pr\big[\,\mathbf{G}_1^0(\mathcal{S}, \mathcal{D})\,\big]$, where $\mathbf{G}_2$ is defined just like $\mathbf{G}_1^0$, but **P** is now as defined in Figure 9. This is because the condition on line 9:4 is never met in case $j = 0$. From a hybrid argument it follows that

$$\Pr\big[\,\mathbf{G}_1^p(\mathcal{S}, \mathcal{A})\,\big] \le \delta P + \Pr\big[\,\mathbf{G}_2(\mathcal{S}, \mathcal{A})\,\big]\,, \qquad (14)$$

where $\delta = \max_{1 \le j \le P} \delta_j$ and $\delta_j = \Big|\Pr\big[\,\mathbf{G}_1^j(\mathcal{S}, \mathcal{A})\,\big] - \Pr\big[\,\mathbf{G}_1^{j-1}(\mathcal{S}, \mathcal{A})\,\big]\Big|$.

Now consider the SEC/I-adversary $\mathcal{B}$ that composes $\mathcal{S}$ and $\mathcal{A}$ in experiment $\mathbf{G}_2$ as follows. Adversary $\mathcal{B}$ declares a set $\mathcal{Q}$, initially empty, and defines the following three oracles. On input of (X, V), oracle $\mathbf{P}'$ runs $\mathbf{R}(X)$. Oracle $\mathbf{Q}'$ just returns the current value of $\mathcal{Q}$. On input of X, oracle $\mathbf{R}'$ runs $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{X\}$ and returns $\mathbf{R}(X)$.

Adversary $\mathcal{B}$ first runs $\sigma \twoheadleftarrow \mathcal{S}.\text{Init}^{\mathbf{P}',\mathbf{Q}',\mathbf{R}}(pk)$, where pk is the public key that it gets as input. It then runs $\langle \mathcal{A} : \mathbf{Init}, \mathbf{Final}, \mathbf{Call}, \mathbf{Op}', \mathbf{R}'\rangle(pk)$, where on input (ctx, op, in), oracle $\mathbf{Op}'$ returns the output of $\mathcal{S}_\sigma{}^{\mathbf{Op},\mathbf{P}',\mathbf{Q}',\mathbf{R}}(ctx, op, in)$. Since $\mathcal{S}$ is regular by assumption, none of its $\mathbf{Op}'$ queries will coincide with the game context. Then $\Pr\big[\,\mathbf{Exp}^{\text{sec/i}}_{\mathcal{I}^0,\mathcal{G}}(\mathcal{B})\,\big] = \Pr\big[\,\mathbf{G}_2(\mathcal{S}, \mathcal{A})\,\big]$ holds by construction, and

$$\Pr\big[\,\mathbf{Exp}^{\text{sec/i}}_{\mathcal{I}^1,\mathcal{G}}(\mathcal{A})\,\big] \le \mathbf{Adv}^{\text{gap2}}_{\mathcal{I}^1,\mathcal{I}^0,\mathcal{G}}(\mathcal{S}, \mathcal{D}) + q_R P / 2^{\mu_1} + \delta P + \Pr\big[\,\mathbf{Exp}^{\text{sec/i}}_{\mathcal{I}^0,\mathcal{G}}(\mathcal{B})\,\big]\,. \qquad (15)$$

Since neighboring hybrids differ only in the distribution of the range value of the j-th point programmed

```
$\mathbf{G}^b_{\mathcal{G}'}(\mathcal{B},\mathcal{D})$
1  dec str $pk, sk, st, \alpha$; bool $win$
2  $(pk, sk) \twoheadleftarrow \mathcal{I}^b.\mathrm{Gen}(\,)$
3  $\langle \mathcal{B}\colon \mathbf{Init}, \mathbf{Final}, \mathbf{Call}, \mathbf{Op}\rangle(pk)$
4  ret $win$

$\mathbf{Init}(ctx)$
5  $(st, out) \twoheadleftarrow \mathcal{G}'.\mathrm{Init}(pk, ctx)$
6  $\alpha \leftarrow ctx$; ret $out$

$\mathbf{Final}(in)$
7  $win \twoheadleftarrow \langle \mathcal{G}'.\mathrm{Final}\colon \mathcal{I}^b_{sk}\rangle(st, in)$
8  $c \twoheadleftarrow \{0,1\}$; $d \twoheadleftarrow \langle \mathcal{D}\colon \mathcal{I}^b_{sk}\rangle(pk)$
9  if $c = 1$ then ret $win \vee (d = 1)$
10 ret $win \wedge (d = 1)$

$\mathbf{Call}(in)$
11 ret $\langle \mathcal{G}'_{st}\colon \mathcal{I}^b_{sk}\rangle(in)$

$\mathbf{Op}(ctx, op, in)$
12 if $ctx = \alpha$ then ret $\bot$
13 ret $\mathcal{I}^b_{sk}(ctx, op, in)$

$\mathcal{G}.\mathrm{Init}(pk, ctx)$
14 $(st', out') \twoheadleftarrow \mathcal{G}'.\mathrm{Init}(pk, ctx)$
15 ret $(\underline{pk, st'}, out')$

$\mathcal{G}.\mathrm{Call}^{\mathcal{O}}(\&st, in)$
16 dec str $pk, st'$
17 $\underline{pk, st'} \leftarrow st$
18 $out \twoheadleftarrow \mathcal{G}'^{\mathcal{O}}_{st'}(in)$
19 $st \leftarrow \underline{pk, st'}$
20 ret $out$

$\mathcal{G}.\mathrm{Final}^{\mathcal{O}}(st, in)$
21 dec str $pk, st'$
22 $\underline{pk, st'} \leftarrow st$
23 $win \twoheadleftarrow \mathcal{G}'.\mathrm{Final}^{\mathcal{O}}(st', in)$
24 $c \twoheadleftarrow \{0,1\}$; $d \twoheadleftarrow \mathcal{D}^{\mathcal{O}}(pk)$
25 if $c = 1$ then ret $win \vee (d = 1)$
26 ret $win \wedge (d = 1)$
```

Figure 10: left: experiment $\mathbf{G}^b_{\mathcal{G}'}(\mathcal{B},\mathcal{D})$; and right: game $\mathcal{G}$ for proof of Theorem 2.

by $\mathbf{P}$, by Lemma 1 we have that

$$\delta_j \leq \frac{1}{2}\sum_{v\in\{0,1\}^h} \big|\Pr\big[\,Y_i = v\,\big] - \Pr\big[\,U_i = v\,\big]\big| \tag{16}$$

$$\leq \frac{1}{2}\sum_{v\in\{0,1\}^h}\left(\frac{1}{2^{\mu_2}} - \frac{1}{2^h}\right) = \frac{2^h}{2^{\mu_2+1}} - \frac{1}{2} = \frac{1}{2^{\mu_2-h+1}} - \frac{1}{2} \tag{17}$$

for each j, where Eq. (17) follows from the assumption that $\mu_2 \leq h$. Note that $P = c_P(q_I + 1) \leq 2c_P q_I$, and so $\delta P \leq c_P q_I (2^{h-\mu_2} - 1)$.

To complete the proof, we need only to account for $\mathcal{D}$ and $\mathcal{B}$'s resources. Both $\mathcal{D}$ and $\mathcal{B}$ make at most q_G queries to **Call**; adversary $\mathcal{D}$ (resp. $\mathcal{B}$) makes at most q_I (resp. $c_I q_I$) queries to **Op**; and both adversary $\mathcal{D}$ and $\mathcal{B}$ make at most $q_R + (c_R + c_P)(q_I + 1) = \hat{q_R}$ queries to $\mathbf{R}$.

Condition (i) follows from essentially the same argument. The GAP1 adversary is identical to $\mathcal{D}$, and the SEC adversary is defined just like $\mathcal{B}$, except that the simulator is given $\bot$ instead of **Op**.

A.2 Theorem 2 (necessity of wGAP2 for Theorem 1(ii))

Define $\mathcal{S}$ as follows: on input of pk, the initiator $\mathcal{S}.\mathrm{Init}$ returns ε; and on input of $(\&\sigma, ctx, op, in)$, the operator $\mathcal{S}.\mathrm{Op}^{\mathcal{O}}$ runs $out \twoheadleftarrow \mathcal{O}(ctx, op, in)$ and returns out. Let d_{xy} denote the probability that $\mathcal{D}$ outputs y in the wGAP2 experiment with simulator $\mathcal{S}$ given that the challenge bit b is equal to x.

Let $\mathcal{A} = \mathcal{B}$. Let $\mathcal{G}'$ be any game and define game $\mathcal{G}$ from $\mathcal{D}$ and $\mathcal{G}'$ as specified in Figure 10. This game acts as a shim for $\mathcal{G}'$, but changes the winning condition as follows. It flips a coin c and runs $d \twoheadleftarrow \mathcal{D}^{\mathcal{O}}(pk)$ (10:24). If $c = 1$ then the outcome is 1 if $\mathcal{G}'$ is in a winning state *or* $d = 1$; if $c = 0$, then the outcome is 1 if $\mathcal{G}'$ is in a winning state *and* $d = 1$.

```
$\mathbf{G}_0(\mathcal{S})$ / $\mathbf{G}_1(\mathcal{S})$
    dec set $\mathcal{R}$; str $W^*, \pi[\,]$; $P \in \mathbb{G}$; $r \in \mathbb{Z}_n$
    $(\underline{P}, sk) \twoheadleftarrow \mathcal{I}.\mathrm{Gen}(\,)$; $r \twoheadleftarrow \mathbb{Z}_n$
    $W^* \twoheadleftarrow \{0,1\}^h$; $\pi[rP, \varepsilon] \leftarrow W^*$; $\mathcal{R} \leftarrow \{W^*\}$
    $\sigma \twoheadleftarrow \langle \mathcal{S}.\mathrm{Init}\colon \mathbf{P}, \mathbf{Q}, \mathbf{R}\rangle(\underline{P})$
    $W \twoheadleftarrow \langle \mathcal{S}_\sigma\colon \mathcal{I}_{sk}, \mathbf{P}, \mathbf{Q}, \mathbf{R}\rangle(\varepsilon, \mathsf{fdh}, \underline{rG})$
    $W^* \twoheadleftarrow \mathbf{R}(rP, \varepsilon)$
    ret $W = W^*$

$\mathbf{P}(X, V)$
    if $X = (rP, \varepsilon)$ then $\pi[X] \leftarrow \diamond$
    $\pi[X] \leftarrow V$

$\mathbf{R}(X)$
    if $X = (rP, \varepsilon)$ then $\pi[X] \leftarrow \diamond$
    $W \twoheadleftarrow \{0,1\}^h$
    if $\pi[X]$ then $W \leftarrow \pi[X]$
    $\pi[X] \leftarrow W$; ret $W$

$\mathbf{Q}(\,)$: ret $\emptyset$
```

```
$\mathbf{P}(X, V)$                                  $\mathbf{G}_1$ $\boxed{\mathbf{G}_2}$
    $\boxed{\text{if } V = W^* \text{ then ret } \bot}$
    $\pi[X] \leftarrow V$

$\mathbf{R}(X)$                                     $\mathbf{G}_2$ $\boxed{\mathbf{G}_3}$
    $W \twoheadleftarrow \{0,1\}^h$ $\boxed{\setminus \mathcal{R}}$
    if $\pi[X]$ then $W \leftarrow \pi[X]$
    $\boxed{\mathcal{R} \leftarrow \mathcal{R} \cup \{W\};}$ $\pi[X] \leftarrow W$; ret $W$

$\mathbf{P}(X, V)$
    if $V = W^*$ then ret $\bot$
    $\boxed{\mathcal{R} \leftarrow \mathcal{R} \cup \{V\};}$ $\pi[X] \leftarrow V$
```

```
$\mathcal{A}^{\mathcal{O}}(P, Q)$
    dec set $\mathcal{R}$; str $\pi[\,]$; elem$_{\mathbb{G}}$ $\rho[\,]$
    $\sigma \twoheadleftarrow \langle \mathcal{S}.\mathrm{Init}\colon \mathbf{P}, \mathbf{Q}, \mathbf{R}\rangle(\underline{P})$
    $W \twoheadleftarrow \langle \mathcal{S}_\sigma\colon \mathcal{O}, \mathbf{P}, \mathbf{Q}, \mathbf{R}\rangle(\varepsilon, \mathsf{fdh}, \underline{Q})$
    ret $\rho[W, \varepsilon]$

$\mathbf{P}((Z, ctx), V)$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{V\}$; $\pi[Z, ctx] \leftarrow V$

$\mathbf{R}(Z, ctx)$
    $W \twoheadleftarrow \{0,1\}^h \setminus \mathcal{R}$
    if $\pi[Z, ctx]$ then $W \leftarrow \pi[Z, ctx]$
    $\rho[W, ctx] \leftarrow Z$; $\pi[Z, ctx] \leftarrow W$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{W\}$; ret $W$

$\mathbf{Q}(\,)$ : ret $\emptyset$
```

Figure 11: Experiments 0 – 3 and IDH-adversary $\mathcal{A}$ for proof of Theorem 4.

Let g_x denote the probability that $\mathbf{G}^x_{\mathcal{G}'}(\mathcal{B}, \mathcal{D}) = 1$, where $\mathbf{G}^x_{\mathcal{G}'}(\mathcal{B}, \mathcal{D})$ is as specified in Figure 10, and let g_{xy} denote the probability that $\mathbf{G}^x_{\mathcal{G}'}(\mathcal{B}, \mathcal{D}) = 1$ given that $c = y$ (see line 10:7). Then

$$\begin{aligned}
g_1 &= 0.5g_{11} + 0.5g_{10} && (18)\\
2g_1 &= g_{11} - g_{01} + g_{01} + g_{10} - g_{00} + g_{00} && (19)\\
&= g_{01} + g_{00} + (g_{11} - g_{01}) + (g_{10} - g_{00}) && (20)\\
&= 2g_0 + (g_{11} - g_{00}) + (g_{10} - g_{01})\,. && (21)
\end{aligned}$$

When $c = 1$, the probability that $\mathcal{B}$ wins is *at least* the probability that $\mathcal{D}$ outputs 1; this implies that $g_{11} \geq d_{11}$ and $g_{01} \leq (1 - d_{00})$. When $c = 0$, the probability that $\mathcal{B}$ wins is *at most* the probability that $\mathcal{D}$ outputs 1; this implies that $g_{00} \leq d_{01}$ and $g_{10} \geq (1 - d_{10})$. It follows that

$$\begin{aligned}
2g_1 &\geq 2g_0 + (d_{11} - d_{01}) + (1 - d_{10}) - (1 - d_{00}) && (22)\\
&= 2g_0 + (d_{11} - d_{01}) + (d_{00} - d_{10}) && (23)\\
g_1 &\geq g_0 + \mathbf{Adv}^{\mathrm{wgap2}}_{\mathcal{I}^1, \mathcal{I}^0}(\mathcal{S}, \mathcal{D})\,. && (24)
\end{aligned}$$

The last equation follows by conditioning on the outcome of the coin flip b in the wGAP2 experiment. Noting that $g_1 = \Pr\big[\,\mathbf{Exp}^{\mathrm{sec/i}}_{\mathcal{I}^1, \mathcal{G}}(\mathcal{A})\,\big]$ and $g_0 = \Pr\big[\,\mathbf{Exp}^{\mathrm{sec/i}}_{\mathcal{I}^0, \mathcal{G}}(\mathcal{B})\,\big]$ completes the proof.

A.3 Theorem 4 (wGAP2 insecurity of functional DH)

Define $\langle \mathcal{D}\colon \mathbf{Op}, \mathbf{R}\rangle(\underline{P})$ as follows. Run $r \twoheadleftarrow \mathbb{Z}_n$, $W \twoheadleftarrow \mathbf{Op}(\varepsilon, \mathsf{fdh}, \underline{rG})$, and $W^* \twoheadleftarrow \mathbf{R}(rP, \varepsilon)$. Return 1 if and only if $W = W^*$. Note that if $b = 1$, then the response will be $\pi[srG] = \pi[rP] = W^*$, and so $\mathcal{D}$ will output 1. Let $\mathbf{G}_0(\mathcal{S})$ be the experiment specified in Figure 11. While the game sets the value $\pi[rP, \varepsilon]$ to W^* before invoking the simulator, if one of the simulator's RO queries coincide with (rP, ε), then the value of $\pi[rP, \varepsilon]$ will be overwritten (line 11:9 and 13). Thus, this experiment is equivalent to running the wGAP2 game with $\mathcal{D}$ and $\mathcal{S}$. Then

$$\mathbf{Adv}^{\mathrm{wgap2}}_{\mathcal{I}_{+\mathrm{fdh}},\mathcal{I}}(\mathcal{S}, \mathcal{D}) = 1 - \Pr\left[\, \mathbf{G}_0(\mathcal{S}) \,\right]. \tag{25}$$

We proceed with a game-playing argument. Our goal is to rewrite the $\mathbf{R}$ and $\mathbf{P}$ oracles so that we can use $\mathcal{S}$ to solve the IDH problem for $(\mathbb{G}, \mathcal{I})$. In particular, we will exhibit a game $\mathbf{G}_3$ in which $\mathcal{S}$ must query $\mathbf{R}$ on the correct point in order to do any better than just guessing W^*. Consider the experiment $\mathbf{G}_1$, also defined in Figure 11. The difference between it and $\mathbf{G}_0$ is that the value of $\pi[rP, \varepsilon]$ is no longer overwritten by $\mathcal{S}$'s RO queries. These experiments are identical as long as none of $\mathcal{S}$'s RO queries coincide with the point rP. Since $|\mathbb{G}| = n$ is prime and $P \in \mathbb{G}$, by Lagrange's Theorem we have that $\mathbb{G} = \langle P \rangle$. Since r is chosen uniform randomly from $\mathbb{Z}_n$, for any query X the probability that $X = (rP, \varepsilon)$ is at most $1/n$. The simulator initiator $\mathcal{S}$.Init and operator $\mathcal{S}$.Op are both run exactly once, and so by the Fundamental Lemma of Game Playing [12] we have that

$$\Pr\left[\, \mathbf{G}_0(\mathcal{S}) \,\right] \le \frac{2(q+p)}{n} + \Pr\left[\, \mathbf{G}_1(\mathcal{S}) \,\right]. \tag{26}$$

Next, $\mathbf{G}_2$ is defined just like $\mathbf{G}_1$, except oracle $\mathbf{P}$ does as follows. On input of $((Z, \mathit{ctx}), V)$, if $V \ne W^*$, then set $\pi[Z, \mathit{ctx}] \leftarrow V$; otherwise do nothing. Since $\mathcal{S}$.Init and $\mathcal{S}$.Op are each executed once, we have that

$$\Pr\left[\, \mathbf{G}_0(\mathcal{S}) \,\right] \le \frac{2(q+p)}{n} + \frac{2p}{2^h} + \Pr\left[\, \mathbf{G}_2(\mathcal{S}) \,\right]. \tag{27}$$

Next, modify $\mathbf{G}_2$ to get a new game, $\mathbf{G}_3$, as follows. The game declares a set $\mathcal{R}$, initially containing just W^*. On input of $((Z, \mathit{ctx}), V)$, if $\mathbf{P}$ sets $\pi[Z, \mathit{ctx}] \leftarrow V$, then it also adds V to the set $\mathcal{R}$. Moreover, on input of (Z, ctx), instead of sampling W from $\{0,1\}^h$, oracle $\mathbf{R}$ samples W from $\{0,1\}^h \setminus \mathcal{R}$. After doing so it adds W to $\mathcal{R}$. Then experiments $\mathbf{G}_2$ and $\mathbf{G}_3$ are identical up to the collision of two or more range points sampled by $\mathbf{R}$ (11:17). This at most $(2p+1)/2^h$ for the first of these queries, since the size of $|\mathcal{R}| \le 2p+1$; for the i-th query, this probability is at most $(2p+i)/2^h$. Summing over all $\mathbf{R}$ queries, we have

$$\begin{aligned} \Pr\left[\, \mathbf{G}_2(\mathcal{S}) \,\right] &\le \Pr\left[\, \mathbf{G}_3(\mathcal{S}) \,\right] + \sum_{i=1}^{2q} \frac{2p+i}{2^h} && (28) \\ &= \Pr\left[\, \mathbf{G}_3(\mathcal{S}) \,\right] + \frac{4pq}{2^h} + \sum_{i=1}^{2q} \frac{i}{2^h} && (29) \\ &\le \Pr\left[\, \mathbf{G}_3(\mathcal{S}) \,\right] + \frac{4pq}{2^h} + \frac{4q^2}{2^h}. && (30) \end{aligned}$$

Combining Eq. (27) with Eq. (30) we obtain

$$\begin{aligned} \Pr\left[\, \mathbf{G}_0(\mathcal{S}) \,\right] &\le \Pr\left[\, \mathbf{G}_3(\mathcal{S}) \,\right] + \frac{2(q+p)}{n} + \frac{2p}{2^h} + \frac{4pq}{2^h} + \frac{4q^2}{2^h} && (31) \\ &\le \Pr\left[\, \mathbf{G}_3(\mathcal{S}) \,\right] + \frac{2(q+p)}{n} + \frac{4p^2 + 4pq + 4q^2}{2^h} && (32) \\ &= \Pr\left[\, \mathbf{G}_3(\mathcal{S}) \,\right] + \frac{\hat{q}}{n} + \frac{\hat{q}^2}{2^h}. && (33) \end{aligned}$$

Now consider the IDH-adversary $\mathcal{A}$ specified in Figure 11. It executes the simulator $\mathcal{S}$ just as in the proof of Theorem 3 except that it must also answer $\mathcal{S}$'s RO queries. It does so using oracles $\mathbf{P}$, $\mathbf{Q}$, and $\mathbf{R}$ specified in the same figure .For each query (Z, ctx) to $\mathbf{R}$ it records the response W in an associative array ρ by setting $\rho[W, \mathit{ctx}] \leftarrow Z$. Since each W is distinct, each element of ρ is set exactly once. When the simulator halts and outputs W, adversary $\mathcal{A}$ halts and outputs $\rho[W, \varepsilon]$. Consider the probability that $\mathbf{G}_3(\mathcal{S}) = 1$ conditioned on the event that $\mathcal{S}$ queries its oracle on rP. If this happens, then $\mathcal{A}$ wins its game, and so the probability that $\mathcal{A}$ wins is at least the probability that the game outputs 1. Now suppose that $\mathcal{S}$ does not guess the point rP; then the probability it manages to output the correct string W^* is at most $1/2^h$. Then $\Pr[\,\mathbf{G}_3(\mathcal{S})\,] \le \mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\mathcal{A}) + 1/2^h$. It follows that

$$\begin{aligned}\Pr[\,\mathbf{G}_0(\mathcal{S})\,] &\le \frac{\hat{q}}{n} + \frac{\hat{q}^2}{2^h} + \frac{1}{2^h} + \mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\mathcal{A}) &(34)\\ &\le \frac{\hat{q}}{n} + \frac{\hat{q}^2}{2^{h-1}} + \mathbf{Adv}^{\mathrm{idh}}_{\mathbb{G},\mathcal{I}}(\mathcal{A})\,, &(35)\end{aligned}$$

which yields the claimed bound.

A.4 Theorem 5 (GAP1 security of EdDSA)

Lemma 2. *For all $Q \in \mathbb{G}$ and $N \ge n$ it holds that $\Pr[\,Q = rG : r \twoheadleftarrow \mathbb{Z}_N\,] \le 2/n$, where $\mathbb{G} = \langle G \rangle$ is an additive group of order n.*

Proof. Let $Q \in \mathbb{G}$ and let $k \ge 1$ be the largest integer such that $kn \le N$. Conditioning on the event that $r < kn$ we have

$$\Pr_r[\,Q = rG\,] = \Pr_r[\,Q = rG \mid r < kn\,]\frac{kn}{N} + \Pr_r[\,Q = rG \mid r \ge kn\,]\frac{N - kn}{N}\,, \tag{36}$$

where $r \twoheadleftarrow \mathbb{Z}_N$. If $r < kn$, then for each $Q \in \mathbb{G}$ the probability that $Q = rG$ is exactly $1/n$. If $r \ge kn$, then rG is one of $N - kn$ equiprobable points in $\mathbb{G}$, and so the probability that $Q = rG$ is at most $1/(N - kn)$. Then

$$\Pr_r[\,Q = rG\,] \le \frac{1}{n} \cdot \frac{kn}{N} + \frac{1}{N - kn} \cdot \frac{N - kn}{N} = \frac{k+1}{N} \le \frac{2k}{N}\,. \tag{37}$$

Since $kn \le N$ it follows that $2k/N \le 2/n$. □

The proof is by a game-playing argument. Let $\mathcal{D}$ be a $\boldsymbol{r}$-resource GAP1-adversary. In Figure 12 we define an experiment $\mathbf{G}_0(\mathcal{S}, \mathcal{D})$, which is similar to $\mathbf{Exp}^{\mathrm{gap1}}_{\mathcal{ED},\mathcal{G}}(\mathcal{S}, \mathcal{D})$, except that it performs some additional book keeping. First, it declares the following additional variables: **str** $X, \rho[]$; **bool** *bad*; and **set** $\mathcal{Y}$. Second, set $\mathcal{Y}$ is populated with each context string $ctx \ne \alpha$ passed to **Op**; if $\mathcal{ED}$ (via $\mathcal{G}$) ever makes an RO query u such that $vr^{\mathbf{R}}(ctx) \preceq u$ for some $ctx \in \mathcal{Y}$, then the flag *bad* gets set. Third, string X is set to a random element of $\{0,1\}^b$ immediately after executing the key generator (and prior to running the simulator or adversary). This will be used as a global variable later on, but for now the value of X has no impact on the outcome of the game. We note that $\mathbf{G}_0(\mathcal{S}, \mathcal{D})$ is $O(t + q_R q_I)$-time since $\mathcal{D}$ is t-time and the bookkeeping overhead for *bad* is $O(q_R q_I)$.

Next, we define an experiment $\mathbf{G}_1$ by modifying $\mathbf{G}_0$ so that, immediately after setting *bad* (line 12:12), oracle $\mathbf{R}'$ halts and returns $\perp$. Because $\mathcal{G}$ is regular, this change ensures that in the new experiment, no queries made by $\mathcal{ED}$ via **Call**, except those involved in computing the scalar s and randomizer X (12:23–24), coincide with those made by Op via **Op**. If a query u to $\mathbf{R}'$ sets *bad*, then this implies that $vr^{\mathbf{R}}(\alpha) = vr^{\mathbf{R}}(ctx)$ for some $ctx \in \mathcal{Y}$ for which $ctx \ne \alpha$. Thus, one can easily exhibit a $O(t + q_R q_I)$-time CR-adversary $\mathcal{C}$ such that

$$\Pr[\,\mathbf{G}_0(\mathcal{S}, \mathcal{D})\,] - \Pr[\,\mathbf{G}_1(\mathcal{S}, \mathcal{D})\,] \le c q_I \mathbf{Adv}^{\mathrm{cr}}_{vr}(\mathcal{C})\,. \tag{38}$$

The adversary executes $\mathbf{G}_0(\mathcal{S}, \mathcal{D})$: if *bad* gets set, then it searches $\mathcal{Y}$ for a string ctx that collides with α under vr and outputs (ctx, α).

We define experiment $\mathbf{G}_2$ from $\mathbf{G}_1$ by modifying the Op procedure called by **Op** as follows: remove line line 12:24 (execution of $X \twoheadleftarrow \mathbf{R}(K)[b+1{:}]$); instead, this procedure will use the X generated at the start of the experiment (12:3). These experiments are identical unless $\mathcal{D}$ manages to ask K of $\mathbf{R}$, and so we can show that

$$\Pr[\,\mathbf{G}_0(\mathcal{S}, \mathcal{D})\,] - \Pr[\,\mathbf{G}_1(\mathcal{S}, \mathcal{D})\,] \le q_R/2^b\,. \tag{39}$$

Next, experiment $\mathbf{G}_3$ is constructed from $\mathbf{G}_2$ as follows (see Figure 12). Replace $\underline{r}_{2b} \twoheadleftarrow \mathbf{R}(A \,\|\, X \,\|\, B)$ on line 12:35 with the following procedure: if $\rho[A, B]$ is undefined, then sample $\rho[A, B] \twoheadleftarrow \{0,1\}^{4b}$; then set $\underline{r}_{2b} \leftarrow$

```
$\mathbf{G}_0(\mathcal{S}, \mathcal{D})$
  $\textbf{dec str}\ K, X, st, \sigma, \alpha, \rho[\,], \pi[\,]$
  $\textbf{dec set}\ \mathcal{Y};\ \textbf{bool}\ c, d, bad;\ P \in \mathbb{G}$
  $(\underline{P}, K) \twoheadleftarrow \mathrm{Gen}^{\mathbf{R}}(\,);\ X \twoheadleftarrow \{0,1\}^b$
  $c \twoheadleftarrow \{0,1\};\ \sigma \twoheadleftarrow \mathcal{S}.\mathrm{Init}^{\mathbf{P},\mathbf{R}}(\underline{P})$
  $d \twoheadleftarrow \langle \mathcal{D}\colon \mathbf{Init}, \mathbf{Final}, \mathbf{Call}, \mathbf{Op}, \mathbf{R} \rangle(\underline{P})$
  $\textbf{ret}\ (d = c)$

$\mathbf{Op}(ctx, op, in)$
  $\textbf{if}\ ctx = \alpha\ \textbf{then ret}\ \perp$
  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{ctx\}$
  $\textbf{if}\ c = 1\ \textbf{then ret}\ \mathrm{Op}^{\mathbf{R}}_K(ctx, op, in)$
  $\textbf{ret}\ \mathcal{S}^{\perp,\mathbf{P},\mathbf{R}}_{\sigma}(ctx, op, in)$

$\mathbf{R}'(u)$
  $\textbf{if}\ (\exists\, ctx \in \mathcal{Y})\ vr^{\mathbf{R}}(ctx) \preceq u\ \textbf{then}$
      $bad \leftarrow 1$
  $\textbf{ret}\ \mathbf{R}(u)$

$\mathbf{P}(u, v)$
  $\pi_u \leftarrow v$

$\mathbf{R}(u)$
  $\textbf{if}\ \neg\pi_u\ \textbf{then}\ \pi_u \twoheadleftarrow \{0,1\}^{2b};\ \textbf{ret}\ \pi_u$
```

```
$\mathbf{Init}(ctx)$
  $(st, out) \twoheadleftarrow \mathcal{G}.\mathrm{Init}^{\mathbf{R}}(pk, ctx)$
  $\alpha \leftarrow ctx;\ \textbf{ret}\ out$

$\mathbf{Final}(in)$
  $\textbf{ret}\ \langle \mathcal{G}.\mathrm{Final}\colon \mathcal{ED}^{\mathbf{R}'}_K, \mathbf{R} \rangle(st, in)$

$\mathbf{Call}(in)$
  $\textbf{ret}\ \langle \mathcal{G}.\mathrm{Call}\colon \mathcal{ED}^{\mathbf{R}'}_K, \mathbf{R} \rangle(\&st, in)$

$\mathrm{Gen}^{\mathbf{R}}(\,)$
  $K \twoheadleftarrow \{0,1\}^b;\ s \twoheadleftarrow cl^{\mathbf{R}}(\mathbf{R}(K)[:b])$
  $\textbf{ret}\ (\underline{sG}, K)$

$\mathrm{Op}^{\mathbf{R}}_K(ctx, op, in)$
  $\textbf{dec}\ r, t \in \mathbb{N}$
  $s \leftarrow cl^{\mathbf{R},\mathbf{R}}(\mathbf{R}(K)[b:])$
  $X \twoheadleftarrow \mathbf{R}(K)[b+1:]$
  $\textbf{if}\ op \neq \mathsf{sig}\ \textbf{then ret}\ \perp$
  $A \leftarrow vr^{\mathbf{R}}(ctx);\ B \leftarrow ph^{\mathbf{R}}(in)$
  $\underline{r}_{2b} \twoheadleftarrow \mathbf{R}(A \,\|\, X \,\|\, B)$
  $\underline{t}_{2b} \twoheadleftarrow \mathbf{R}(A \,\|\, \underline{rG} \,\|\, \underline{P} \,\|\, B)$
  $x \leftarrow r + st \pmod{n};\ R \leftarrow rG$
  $\textbf{ret}\ \underline{R, x}$
```

```
$\mathrm{Op}(sk, ctx, op, in)$    $\mathbf{G}_2\ \boxed{\mathbf{G}_3}$
  $\textbf{dec}\ r, t \in \mathbb{N}$
  $s \leftarrow cl^{\mathbf{R}}(\mathbf{R}(K)[b:])$
  $\textbf{if}\ op \neq \mathsf{sig}\ \textbf{then ret}\ \perp$
  $A \leftarrow vr^{\mathbf{R}}(ctx);\ B \leftarrow ph^{\mathbf{R}}(in)$
  $\underline{r}_{2b} \twoheadleftarrow \mathbf{R}(A \,\|\, X \,\|\, B)$
  $\textbf{if}\ \rho_{A,B} = \diamond\ \textbf{then}$
      $\rho_{A,B} \twoheadleftarrow \{0,1\}^{4b}$
  $\underline{r}_{2b} \leftarrow \rho_{A,B}[:2b];\ \underline{t}_{2b} \leftarrow \rho_{A,B}[2b+1:]$
  $x \leftarrow r \pmod{n}$
  $\underline{t}_{2b} \twoheadleftarrow \mathbf{R}(A \,\|\, \underline{rG} \,\|\, \underline{P} \,\|\, B)$
  $x \leftarrow r + st \pmod{n};\ R \leftarrow rG$
  $\textbf{ret}\ \underline{R, x}$
```

```
$\mathrm{Op}(sk, ctx, op, in)$    $\mathbf{G}_3\ \boxed{\mathbf{G}_4}$
  $\textbf{dec}\ r, t \in \mathbb{N}$
  $s \leftarrow cl^{\mathbf{R}}(\mathbf{R}(K)[b:])$
  $\textbf{if}\ op \neq \mathsf{sig}\ \textbf{then ret}\ \perp$
  $A \leftarrow vr^{\mathbf{R}}(ctx);\ B \leftarrow ph^{\mathbf{R}}(in)$
  $\textbf{if}\ \rho_{A,B} = \diamond\ \textbf{then}$
      $\rho_{A,B} \twoheadleftarrow \{0,1\}^{4b}$
  $\underline{r}_{2b} \leftarrow \rho_{A,B}[:2b];\ \underline{t}_{2b} \leftarrow \rho_{A,B}[2b+1:]$
  $x \leftarrow r \pmod{n}$
  $\underline{t}_{2b} \twoheadleftarrow \mathbf{R}(A \,\|\, \underline{rG} \,\|\, \underline{P} \,\|\, B)$
  $x \leftarrow r + st \pmod{n};\ R \leftarrow rG$
  $\boxed{\mathbf{P}(A \,\|\, \underline{R} \,\|\, \underline{P} \,\|\, B, \underline{t}_{2b})}$
  $\textbf{ret}\ \underline{R, x}$
```

```
$\mathrm{Op}(sk, ctx, op, in)$    $\mathbf{G}_4\ \boxed{\mathbf{G}_5}$
  $\textbf{dec}\ r, t \in \mathbb{N}$
  $s \leftarrow cl^{\mathbf{R}}(\mathbf{R}(K)[b:]);$
  $\textbf{if}\ op \neq \mathsf{sig}\ \textbf{then ret}\ \perp$
  $A \leftarrow vr^{\mathbf{R}}(ctx);\ B \leftarrow ph^{\mathbf{R}}(in)$
  $\textbf{if}\ \rho_{A,B} = \diamond\ \textbf{then}$
      $\rho_{A,B} \twoheadleftarrow \{0,1\}^{4b}$
  $\underline{r}_{2b} \leftarrow \rho_{A,B}[:2b];\ \underline{t}_{2b} \leftarrow \rho_{A,B}[2b+1:]$
  $x \leftarrow r \pmod{n}$
  $x \leftarrow r + st \pmod{n};\ R \leftarrow rG$
  $\boxed{R \leftarrow xG - tP}$
  $\mathbf{P}(A \,\|\, \underline{R} \,\|\, \underline{P} \,\|\, B, \underline{t}_{2b})$
  $\textbf{ret}\ \underline{R, x}$
```

```
$\mathcal{S}^{\mathrm{ed}}.\mathrm{Init}^{\mathbf{P},\mathbf{R}}(pk)$
  $\textbf{dec str}\ \rho[\,];\ P \in \mathbb{G}$
  $\underline{P} \leftarrow pk;\ \textbf{ret}\ \underline{P, \rho}$

$\mathcal{S}^{\mathrm{ed}}.\mathrm{Op}^{\mathcal{O},\mathbf{P},\mathbf{R}}(\&\sigma, ctx, op, in)$
  $\textbf{dec str}\ \rho[\,];\ \textbf{bool}\ a;\ P \in \mathbb{G};\ x, t \in \mathbb{N}$
  $\textbf{if}\ op \neq \mathsf{sig}\ \textbf{then ret}\ \mathcal{O}(ctx, op, in)$
  $\underline{P, \rho} \leftarrow \sigma;\ A \leftarrow vr^{\mathbf{R}}(ctx);\ B \leftarrow ph^{\mathbf{R}}(in)$
  $\textbf{if}\ \rho_{A,B} = \diamond\ \textbf{then}\ a \leftarrow 1$
      $\rho_{A,B} \twoheadleftarrow \{0,1\}^{4b};\ \sigma \leftarrow \underline{P, \rho}$
  $\underline{x}_{2b} \leftarrow \rho_{A,B}[:2b];\ \underline{t}_{2b} \leftarrow \rho_{A,B}[2b+1:]$
  $x \leftarrow x \pmod{n};\ R \leftarrow xG - tP$
  $\textbf{if}\ a = 1\ \textbf{then}\ \mathbf{P}(A \,\|\, \underline{R} \,\|\, \underline{P} \,\|\, B, \underline{t}_{2b})$
  $\textbf{ret}\ \underline{R, x}$
```

Figure 12: Experiment 0, and revisions for 4, 4, and 5, and **P**-relative simulator $\mathcal{S}^{\mathrm{ed}}$ for proof of Theorem 5.

$\rho[A,B][{:}2b]$. (We also set $\underline{t}_{2b} \leftarrow \rho[A,B][2b+1{:}]$ and $x \leftarrow r \pmod{n}$, but these variables get overwritten immediately thereafter, and so these statements have no affect on the outcome of the experiment.) Since $\mathcal{G}$ is $\mathcal{ED}$-regular, experiment $\mathbf{G}_3$ is identical to $\mathbf{G}_2$ unless $\mathcal{D}$ manages to ask $A \,\|\, X \,\|\, B$ of $\mathbf{R}$, where $A = vr^{\mathbf{R}}(ctx)$ and $B = ph^{\mathbf{R}}(in)$ coincide with any one of its sig-operator queries. Hence,

$$\Pr\left[\,\mathbf{G}_1(\mathcal{S},\mathcal{D})\,\right] - \Pr\left[\,\mathbf{G}_2(\mathcal{S},\mathcal{D})\,\right] \leq q_R q_I / 2^b \,. \tag{40}$$

Next, experiment $\mathbf{G}_4$ is defined from $\mathbf{G}_3$ by modifying Op as follows: remove $\underline{t}_{2b} \twoheadleftarrow \mathbf{R}(A \,\|\, \underline{rG} \,\|\, \underline{P} \,\|\, B)$ on line 12:51 and add the statement $\mathbf{P}(A \,\|\, \underline{R} \,\|\, \underline{P} \,\|\, B, \underline{t}_{2b})$ just prior to the return statement. These experiments are identical unless $\mathcal{D}$ manages to guess an input $A \,\|\, \underline{rG} \,\|\, \underline{P} \,\|\, B$ to $\mathbf{R}$ that coincides with one of its sig-operator queries. To bound this probability, first consider the probability that $Q = rG$ for any $Q \in \mathbb{G}$ and a randomly sampled $\underline{r}_{2b} \twoheadleftarrow \{0,1\}^{2b}$. Note that sampling r in this way is equivalent to sampling r as $r \twoheadleftarrow \mathbb{Z}_{2^{2b}}$ (cf. Section 2). Lemma 2 implies that this probability is at most $2/n$, and so

$$\Pr\left[\,\mathbf{G}_2(\mathcal{S},\mathcal{D})\,\right] - \Pr\left[\,\mathbf{G}_3(\mathcal{S},\mathcal{D})\,\right] \leq 2 q_R q_I / n \,. \tag{41}$$

Finally, in $\mathbf{G}_5$, instead of returning $R = rG$ and $x \equiv r + st \pmod{n}$, we return $R = xG - tP$ and $x \equiv r \pmod{n}$. This does not change the outcome of the experiment, however, since R and x have the same relationship in both experiments (namely, $xG = R + tP$). Observe that the behavior of the $\mathbf{Op}$ in case $b = 1$ is the same as when $b = 0$, and so $\Pr\left[\,\mathbf{G}_5(\mathcal{S},\mathcal{D})\,\right] = 1/2$. Summarizing, we have that

$$\begin{aligned}
\Pr\left[\,\mathbf{Exp}^{\mathrm{gap1}}_{\mathcal{ED},\mathcal{G}}(\mathcal{S},\mathcal{D}\,\right] &\leq c q_I \mathbf{Adv}^{\mathrm{cr}}_{vr}(\mathcal{C}) + \Pr\left[\,\mathbf{G}_1(\mathcal{S},\mathcal{D})\,\right] && (42)\\
&\leq c q_I \mathbf{Adv}^{\mathrm{cr}}_{vr}(\mathcal{C}) + \frac{q_R}{2^b} + \Pr\left[\,\mathbf{G}_2(\mathcal{S},\mathcal{D})\,\right] && (43)\\
&\leq c q_I \mathbf{Adv}^{\mathrm{cr}}_{vr}(\mathcal{C}) + \frac{q_R}{2^b} + \frac{q_R q_I}{2^b} + \Pr\left[\,\mathbf{G}_3(\mathcal{S},\mathcal{D})\,\right] && (44)\\
&\leq c q_I \mathbf{Adv}^{\mathrm{cr}}_{vr}(\mathcal{C}) + \frac{q_R}{2^b} + \frac{q_R q_I}{2^b} + \frac{2 q_R q_I}{n} + \Pr\left[\,\mathbf{G}_4(\mathcal{S},\mathcal{D})\,\right] && (45)\\
&\leq c q_I \mathbf{Adv}^{\mathrm{cr}}_{vr}(\mathcal{C}) + \frac{3 q_R q_I}{n} + \Pr\left[\,\mathbf{G}_5(\mathcal{S},\mathcal{D})\,\right] && (46)\\
&\leq c q_I \mathbf{Adv}^{\mathrm{cr}}_{vr}(\mathcal{C}) + \frac{3 q_R q_I}{n} + \frac{1}{2}\,, && (47)
\end{aligned}$$

where Eq. (46) follows from our assumption that $n \leq 2^{b-1}$. The claimed bound follows by applying the definition of $\mathcal{D}$'s GAP1 advantage.

Note that $\mathcal{S}$ is $O(t/q_I)$-time since, by convention (see Section 2), adversary $\mathcal{D}$'s runtime includes the time required to evaluate its queries, It is $(\log n/2, 2b)$-min-entropy because each query to $\mathbf{P}$ is of the form $(A \,\|\, \underline{xG - tP} \,\|\, \underline{P} \,\|\, B, \underline{t}_{2b})$, where x and t are random elements of $\mathbb{Z}_{2b}$; by Lemma 2 the probability that any Q is equal to $xG - tP$ is at most $2/n = 1/2^{\log n/2}$.

A.5 Theorem 6 (GAP2 security of EdDSA)

The proof is closely related to Theorem 5, except we needn't rely on the collision resistance of vr for context separation. In Theorem 5, this property was used to bound the probability of an adversary distinguishing between an experiment in which the random scalar r is generated honestly and an experiment in which it is computed using a table ρ that is independent of $X = \mathbf{R}(K)$. (See the middle-left panel of Figure 12.) Since $\mathcal{I}$ is simple, it is independent of X by definition, and so we can instead use the unpredictability of X to bound this probability.

Let $\mathcal{D}$ be a $\boldsymbol{r}$-resource GAP2-adversary. We begin with an experiment $\mathbf{G}_0(\mathcal{S},\mathcal{D})$ (Figure 12) that is similar $\mathbf{Exp}^{\mathrm{gap2}}_{\mathcal{ED}+\mathcal{I},\mathcal{I},\mathcal{G}}(\mathcal{S},\mathcal{D})$, but with some changes that will help clarify our argument. First, before executing $\mathcal{D}$, we explicitly compute the public and secret keys. This does not change semantics of the experiment. Second, only the simulator and $\mathcal{ED}$ have direct access to $\mathbf{R}$; all other algorithms' RO queries are proxied by an oracle $\mathbf{R}'$ that checks if $u \sim X$, i.e., if X is non-empty and is a sub-string of u. If so, then it sets a flag *bad*. Note that $\mathbf{G}_0(\mathcal{S},\mathcal{D})$ is $O(t)$-time if $\mathcal{D}$ is t-time.

The first change reflects the second change in the proof of Theorem 5. Define experiment $\mathbf{G}_1$ from $\mathbf{G}_0$ as follows. Modify the call to $\mathcal{ED}$ in $\mathbf{Op}$ with the procedure Op defined in the top panel of Figure 12. Now,

```
$\mathbf{G}_0(\mathcal{S},\mathcal{D})$
  dec str $K, X, st, \sigma, \alpha, \pi[\,]$
  dec bool $c, d, bad$; $P \in \mathbb{G}$; $s \in \mathbb{N}$
  $c \twoheadleftarrow \{0,1\}$; $K \twoheadleftarrow \{0,1\}^b$; $\pi[K] \twoheadleftarrow \{0,1\}^{2b}$
  $s \leftarrow cl(\pi(K)[:b])$
  $P \leftarrow sG$; $\sigma \twoheadleftarrow \mathcal{S}.\mathrm{Init}^{\mathbf{P},\mathbf{R}}(\underline{P})$
  $d \twoheadleftarrow \langle \mathcal{D}\colon \mathbf{Init}, \mathbf{Final}, \mathbf{Call}, \mathbf{Op}, \mathbf{R}_1 \rangle(\underline{P})$
  ret $(d = c)$

$\mathbf{Init}(ctx)$
  $(st, out) \twoheadleftarrow \mathcal{G}.\mathrm{Init}^{\mathbf{R}_2}(pk, ctx)$
  $\alpha \leftarrow ctx$; ret $out$

$\mathbf{Final}(in)$
  ret $\langle \mathcal{G}.\mathrm{Final}\colon \mathbf{I}_c, \mathbf{R}_2 \rangle(st, in)$

$\mathbf{Call}(in)$
  ret $\langle \mathcal{G}.\mathrm{Call}\colon \mathbf{I}_c, \mathbf{R}_2 \rangle(\&st, in)$

$\mathbf{Op}(ctx, op, in)$
  if $ctx = \alpha$ then ret $\bot$
  if $c = 1$ then ret $\mathbf{I}^1(ctx, op, in)$
  ret $\langle \mathcal{S}_\sigma\colon \mathbf{I}_0, \mathbf{P}, \mathbf{R} \rangle(ctx, op, in)$

$\mathbf{I}_x(ctx, op, in)$
  if $x = 1 \wedge op = \mathsf{sig}$ then
    ret $\mathcal{ED}^{\mathbf{R}}_K(ctx, op, in)$
  ret $\mathcal{I}^{\mathbf{R}_2}_{\underline{s}}(ctx, op, in)$

$\mathbf{R}'(u)$
  if $u \sim X$ then $bad \leftarrow 1$
  ret $\mathbf{R}(u)$

$\mathbf{P}(u, v)$
  $\pi_u \leftarrow v$

$\mathbf{R}(u)$
  if $\neg\pi_u$ then $\pi_u \twoheadleftarrow \{0,1\}^{2b}$; ret $\pi_u$
```

Figure 13: Experiments 0 and 1 for proof of Theorem 5.

remove the statement $X \twoheadleftarrow \mathbf{R}(K)[b+1{:}]$ on line 12:24 and add the statement $X \twoheadleftarrow \{0,1\}^b$ just prior to running the adversary. Then

$$\Pr\big[\,\mathbf{G}_0(\mathcal{S},\mathcal{D})\,\big] - \Pr\big[\,\mathbf{G}_1(\mathcal{S},\mathcal{D})\,\big] \le q_R/2^b\,. \tag{48}$$

Next, we construct $\mathbf{G}_2$ from $\mathbf{G}_1$ by modifying $\mathbf{R}'$ so that the oracle halts and outputs $\bot$ immediately after bad gets set. The probability that any query to $\mathbf{R}'$ for any x sets bad_1 is at most $1/2^{|X|} = 1/2^b$. Since the number of queries is at most q_R it follows that

$$\Pr\big[\,\mathbf{G}_1(\mathcal{S},\mathcal{D})\,\big] - \Pr\big[\,\mathbf{G}_2(\mathcal{S},\mathcal{D})\,\big] \le \Pr\big[\,\mathbf{G}_1(\mathcal{S},\mathcal{D})\,\mathbf{sets}\ bad_1\,\big] \le q_R/2^b\,. \tag{49}$$

To complete the proof, we can apply the same rewrites in the bottom panel of Figure 12 to obtain an experiment $\mathbf{G}_3$ such that

$$\Pr\big[\,\mathbf{G}_0(\mathcal{S},\mathcal{D})\,\big] \le 2q_R/2^b + q_Rq_I/2^b + 2q_Rq_I/n + \Pr\big[\,\mathbf{G}_3(\mathcal{S},\mathcal{D})\,\big] \tag{50}$$

and in which $\mathcal{D}$ has no advantage. The only difference is that it uses $\mathcal{I}$ to answer non-sig-operator queries, rather than return nothing (12:57). We have that $\Pr\big[\,\mathbf{G}_2(\mathcal{S},\mathcal{D})\,\big] = 1/2$ since this is precisely the behavior of $\mathcal{S}$. Then

$$\begin{aligned}
\Pr\big[\,\mathbf{G}_0(\mathcal{S},\mathcal{D})\,\big] &\le 2q_R/2^b + q_Rq_I/2^b + 2q_Rq_I/n + 1/2 && (51)\\
\mathbf{Adv}^{\mathrm{gap2}}_{\mathcal{ED}+\mathcal{I},\mathcal{I},\mathcal{G}}(\mathcal{S},\mathcal{D}) &\le 4q_R/2^b + 2q_Rq_I/2^b + 4q_Rq_I/n && (52)\\
&\le 3q_Rq_I/2^{b-1} + 4q_Rq_I/n \le 7q_Rq_I/n\,, && (53)
\end{aligned}$$

yielding the desired bound.

A.6 Theorem 7 (GAP1 security of Noise)

The proof is by a game-playing argument in which we rewrite the **Op** oracle so that its output is independent of the challenge bit b. Our strategy is to change the pseudocode executed by **Op** when $b = 1$ so that it is functionally equivalent to the simulator $\mathcal{S}$ indicated in the theorem statement. We will specify $\mathcal{S}$ at the very end; for now, let $\mathcal{S}$ be any **DDH**- and **Q**-relative simulator. Let $\mathcal{D}$ be a $\boldsymbol{r}$-resource GAP1-adversary.

Consider the experiment $\mathbf{G}_0(\mathcal{S},\mathcal{D})$ defined in Figure 14. This game is similar to $\mathbf{Exp}^{\mathrm{gap1}}_{\mathcal{N},\mathcal{G}}(\mathcal{S},\mathcal{D})$, except it performs some additional bookkeeping. In particular, it declares a variable g of type **sst** (defined in

```
$\mathbf{G}_0(\mathcal{S},\mathcal{D})$ / $\boxed{\mathbf{G}_1(\mathcal{S},\mathcal{D})}$
  dec str $st, \sigma, \alpha$, str $\pi[\,]$; bool $b, d$
  dec set $\mathcal{Q}, \mathcal{Y}$; sst $g$; $s \in \mathbb{N}$
  $(\underline{g.P}, \underline{g.s}) \twoheadleftarrow \mathcal{N}.\mathrm{Gen}(\,)$; $s \leftarrow g.s$
  $\sigma \twoheadleftarrow \mathcal{S}.\mathrm{Init}^{\mathbf{DDH},\mathbf{Q},\mathbf{R}}(\underline{g.P})$; $b \twoheadleftarrow \{0,1\}$
  $d \twoheadleftarrow \langle \mathcal{D}\colon \mathbf{Init}, \mathbf{Final}, \mathbf{Call}, \mathbf{Op}, \mathbf{R}_1 \rangle(\underline{g.P})$
  ret $(d = b)$

$\mathbf{Init}(ctx)$
  $(st, out) \twoheadleftarrow \mathcal{G}.\mathrm{Init}^{\mathbf{R}_2}(\underline{g.P}, ctx)$
  $\alpha \leftarrow ctx$; ret $out$

$\mathbf{Final}(in)$
  ret $\langle \mathcal{G}.\mathrm{Final}\colon \mathcal{N}_{\underline{s}}^{\mathbf{R}_2}, \mathbf{R}_2 \rangle(st, in)$

$\mathbf{Call}(in)$
  ret $\langle \mathcal{G}.\mathrm{Call}\colon \mathcal{N}_{\underline{s}}^{\mathbf{R}_2}, \mathbf{R}_2 \rangle(\&st, in)$

$\mathbf{Op}(ctx, op, in)$
  if $ctx = \alpha$ then ret $\bot$
  $g.known \leftarrow 1$; $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{ctx\}$
  for each $(id, u, v) \in \mathcal{Q} \setminus g.\mathcal{Q}$ do $\underline{Z} \leftarrow v$
    $g.\mathcal{U} \leftarrow g.\mathcal{U} \cup \{(id, u)\}$
    if $Z$ then $g.\mathcal{V} \leftarrow g.\mathcal{V} \cup \{Z\}$
  $g.\mathcal{Q} \leftarrow g.\mathcal{Q} \cup \mathcal{Q}$
  if $b = 1$ then ret $\mathcal{N}_{\underline{s}}^{\mathbf{R}}(ctx, op, in)$
  ret $\mathcal{S}_{\sigma}^{\bot,\mathbf{DDH},\mathbf{Q},\mathbf{R}}(ctx, op, in)$

$\mathbf{R}_c(id, u, v)$
  if $c = 1$ then  // caller is $\mathcal{D}$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(id, u, v)\}$
  if $c = 2$ then  // caller is $\mathcal{G}$ (or $\mathcal{N}$ via $\mathcal{G}$)
    if $(\exists\, ctx \in \mathcal{Y})\, \psi(ctx, (id, u, v))$ then
      $g.bad_1 \leftarrow 1$; $\boxed{\mathbf{ret}\,\bot}$
  ret $\mathbf{R}(id, u, v)$

$\mathbf{R}(id, u, v)$
  if $\neg\pi_{id,u,v}$ then $\pi_{id,u,v} \twoheadleftarrow (\{0,1\}^h)^3$
  ret $\pi_{id,u,v}$

$\mathbf{Q}(\,)$
  ret $\mathcal{Q}$
```

```
dec struct {
  tup $\pi[\,]$; elem$_{\mathbb{G}}$ $\rho[\,]$; set $\mathcal{K}[\,], \mathcal{Q}, \mathcal{R}, \mathcal{U}, \mathcal{V}$;
  bool $known, bad[\,]$; $P \in \mathbb{G}$; $s \in \mathbb{Z}_n$ } sst
```

```
SimCap(bool $f$, &sst $g$, &st $hs$, str $X$)                              $\mathbf{G}_2$
  if $\neg hs.K$ then ret $(X, \mathrm{mHash}(\&hs, X))$
  $(K, A) \leftarrow (hs.K, hs.A)$; $N \leftarrow \underline{hs.seq}_{n'}$
  if $f$ then $Y \leftarrow \mathcal{AE}.\mathrm{Enc}(K, N, A, X)$
  else $Y \leftarrow \mathcal{AE}.\mathrm{Dec}(K, N, A, X)$
  if $Y = \bot$ then ret $(\diamond, \mathsf{err\_cap})$
  if $\neg g.known$ then $g.bad_2 \leftarrow 1$
  $\mathrm{mHash}(\&hs, Y)$; ret $(Y, \mathrm{iNonce}(\&hs))$

$\mathrm{mKeySimPSK}^{\mathbf{R}}$(&sst $g$, &st $hs$, str $psk$)
  if $\neg hs.psk$ then ret $\mathsf{err\_psk}$
  $g.known \leftarrow ((hs.id, hs.L) \in g.\mathcal{U})$
  $(hs.L, L', L'') \twoheadleftarrow \mathbf{R}(hs.id, hs.L, psk)$
  $hs.K \leftarrow L''[{:}k]$; $hs.seq \leftarrow 0$
  $x \leftarrow g.known$; $g.\mathcal{K}_x \leftarrow g.\mathcal{K}_x \cup \{hs.K\}$
  ret $\mathrm{mHash}(\&hs, A)$

$\mathrm{mKeySimDH}^{\mathbf{DDH},\mathbf{R}}$(&sst $g$, &st $hs$, elem$_{\mathbb{G}}$ $Y$)
  dec str $L', L''$; $Z \in \mathbb{G}$
  if $\neg Y$ then ret $\mathsf{err\_dh}$
  $Z \twoheadleftarrow \mathrm{SimDH}^{\mathbf{DDH},\mathbf{R}}(\&g, \&hs, Y)$
  $g.known \leftarrow (Z \wedge (hs.id, hs.L) \in g.\mathcal{U})$
  if $g.known$ then $(hs.L, L', L'') \twoheadleftarrow \mathbf{R}(hs.id, hs.L, \underline{Z})$
  else $(hs.L, L', L'') \twoheadleftarrow \mathbf{R}(hs.id, hs.L, \underline{(g.s)Y})$
  $hs.K \leftarrow L'[{:}k]$; $hs.seq \leftarrow 0$
  $x \leftarrow g.known$; $g.\mathcal{K}_x \leftarrow g.\mathcal{K}_x \cup \{hs.K\}$
  ret $\diamond$

$\mathrm{mKeySimTok}^{\mathbf{DDH},\mathbf{R}}$(&sst $g$, &st $hs$, bool $f, r$, str $t$)
  switch $(f, r, t)$  // “$*$” means “match any”
    case $(0, *, \mathsf{psk})$:
      ret $\mathrm{mKeySimPSK}^{\mathbf{R}}(\&hs, hs.psk)$
    case $(0, 0, \mathsf{se})$, $(0, 1, \mathsf{es})$, $(0, *, \mathsf{ss})$ :
      ret $\mathrm{mKeySimDH}^{\mathbf{DDH},\mathbf{R}}(\&g, \&hs, hs.Q)$
    case $(0, 0, \mathsf{es})$, $(0, 1, \mathsf{se})$, $(0, *, \mathsf{ee})$:
      ret $\mathrm{mKeySimDH}^{\mathbf{DDH},\mathbf{R}}(\&g, \&hs, hs.R)$
    case $(1, *, \mathsf{psk})$:
      ret $\mathrm{mKeyPSK}^{\mathbf{R}}(\&hs, hs.psk)$
    case $(1, *, \mathsf{ee})$:
      ret $\mathrm{mKeyDH}^{\mathbf{R}}(\&hs, hs.e, hs.R)$
    case $(1, 0, \mathsf{se})$, $(1, 1, \mathsf{es})$:
      ret $\mathrm{mKeyDH}^{\mathbf{R}}(\&hs, hs.e, hs.Q)$
    case $(1, 0, \mathsf{es})$, $(1, 1, \mathsf{se})$, $(1, *, \mathsf{ss})$:
      ret $\bot$ // excluded by $\mathcal{T}$
  ret $\mathsf{err\_token}$
```

Figure 14: Experiments 0 — 2 and definition of type **sst** (bottom-left) for proof of Theorem 7.

the bottom-left panel of Figure 14), which we will refer to as the game's "shadow state". First, when the key generator is executed, its output is assigned to $(g.P, g.s)$. Second, the experiment sets a flag $g.bad_1$ if the game $\mathcal{G}$ (or $\mathcal{N}$ via $\mathcal{G}$) ever makes an RO query (id, u, v) that satisfies $\psi(ctx, (id, u, v))$ for some context string $ctx \neq \alpha$ used in a query to **Op**. Third, each time it is executed, the **Op** oracle processes the adversary's fresh RO queries as follows: for each $(u, v) \in g.\mathcal{Q} \setminus \mathcal{Q}$, it adds u to the set $g.\mathcal{U}$, and if v encodes a valid point $Z \in \mathbb{G}$, it adds Z to the set $g.\mathcal{V}$. Finally, it adds $\mathcal{Q}$ to the set $g.\mathcal{Q}$. One can easily verify that none of these changes affect the outcome of the experiment; note, however, that $\mathbf{G}_0(\mathcal{S}, \mathcal{D})$ is $O(t + q_R q_I)$-time since $\mathcal{D}$ is t-time and the bookkeeping overhead is $O(q_R q_I)$.

Experiment $\mathbf{G}_1$ is also specified in Figure 14. It is identical to $\mathbf{G}_0$ until $g.bad_1$ gets set, after which point $\mathbf{R}_2$ returns $\perp$. By the (ψ-ro-)regularity of $\mathcal{G}$ and the definition of $\mathcal{N}$, and by assumption that each of $\mathcal{N}$'s constituents is 0-ro-bound, this implies that $vr(ctx) = vr(\alpha)$. But since $ctx \neq \alpha$, we can bound the probability that flag $g.bad_1$ gets using the collision resistance of vr. In particular, one can easily exhibit a $O(t + q_R q_I)$-time CR-adversary $\mathcal{C}$ such that

$$\Pr\big[\, \mathbf{G}_0(\mathcal{S}, \mathcal{D}) \,\big] - \Pr\big[\, \mathbf{G}_1(\mathcal{S}, \mathcal{D}) \,\big] \leq c q_I \mathbf{Adv}^{\mathrm{cr}}_{vr}(\mathcal{C})\,. \tag{54}$$

The adversary executes $\mathbf{G}_0(\mathcal{S}, \mathcal{D})$: if $g.bad_1$ is set, then it searches $\mathcal{Y}$ for a string ctx that collides with α under vr and outputs (ctx, α).

The next transition rewrites the call to $\mathcal{N}$ made by **Op** in order to set up the reduction to the INT-CTXT security of $\mathcal{AE}$. Consider procedures SimCap, mKeySimPSK, mKeySimDH, and mKeySimTok defined in the top-right panel of Figure 14. These are variants of Cap, mKeyPSK, mKeyDH, and mKeyTok (Figure 8) respectively that use the shadow state instead of the static key. We first describe each procedure's functionality, then specify the changes made in experiment $\mathbf{G}_2$.

SimCap works precisely like Cap, except it does some additional bookkeeping. In particular, if encryption or decryption succeeds and a flag $g.known$ is *not* set, then it sets a another flag $g.bad_2$. Flag $g.known$ is set or unset as defined below.

mKeySimPSK works precisely like mKeyPSK, except it performs some additional bookkeeping. In particular, if the first input to the RO $u = hs.id \,\|\, hs.L$ is in the set $g.\mathcal{U}$, then it sets $g.known$; otherwise it unsets $g.known$ (i.e., it runs $g.known \leftarrow 0$). After the symmetric key state and key are updated (14:40–41) it adds the new key K to a set $g.\mathcal{K}_x$, where $x = g.known$. If $K \in g.\mathcal{K}_1$, then we will say that K is (possibly) known to the adversary; otherwise will say that K is (certainly) unknown.

mKeySimDH is called when processing a key update involving the static secret key. It first attempts to perform the update without using static secret key by invoking a procedure SimDH (14:46) with access to **DDH** and **R** and on input of the shadow state, the handshake state, and the input point Y. (We will define SimDH in a moment.) Let Z denote the output, and let $u = hs.id \,\|\, hs.L$. If $Z \in \mathbb{G}$ (i.e., $Z \neq \diamond$) and $u \in g.\mathcal{U}$, then the procedure sets $g.known$ and unsets it otherwise. Next, if $g.known$ is set, then the state is updated using Z (the output of SimDH); otherwise it uses $(g.s)Y$. Finally, the new key is added to the set $g.\mathcal{K}_x$, where $x = g.known$. Procedure SimDH uses the **DDH** oracle as defined in Figure 15. On input of Y, it updates the shadow state as follows. For each $R \in g.\mathcal{R}$ and $W \in g.\mathcal{V}$, if $g.\rho_R$ is undefined, then it asks $r \leftarrow \mathbf{DDH}(g.P, R, W)$: if $r = 1$, then it sets $g.\rho_R \leftarrow W$. After updating the shadow state, SimDH halts and outputs $g.\rho_Y$, which is either a point in $\mathbb{G}$ or equal to $\diamond$.

Lastly, mKeySimTok is much like mKeyTok, except that it operates on token actions rather than tokens so that it can explicitly disallow the write actions excluded by $\mathcal{T}$ (i.e., the set $\mathcal{X}$). Write actions permitted by $\mathcal{T}$ are processed with the normal procedures (14:60–67), but read actions are processed using their simulated counterparts (54–59).

Now that we have defined the low-level simulation procedures, we are ready to specify the changes to the experiment. Experiment $\mathbf{G}_2$ is defined from $\mathbf{G}_1$ by replacing pseudocode with calls to the procedures defined in Figure 14, as well as variants of the higher-level procedures that we will define.

1. Define wSimTok from wTok as follows: (a) replace execution of $\text{mKeyTok}^{\mathbf{R}}(\&hs, s, r, t)$ with execution of $\text{mKeySimTok}^{\mathbf{R}}(\&g, \&hs, 1, r, t)$; (b) replace $\text{Cap}_1(\&hs, \underline{sG})$ with $\text{SimCap}_1(\&g, \&hs, \underline{g.P})$; and (c) change the signature by removing $\textbf{int}\; s$ and adding $\&\textbf{sst}\; g$ as the first argument.

2. Define rSimTok from rTok in kind: (a) replace execution of $\text{mKeyTok}^{\mathbf{R}}(\&hs, s, r, t)$ with execution of $\text{mKeySimTok}^{\mathbf{DDH},\mathbf{R}}(\&g, \&hs, 0, r, t)$; (b) replace $\text{Cap}_0(\&hs, \underline{sG})$ with $\text{SimCap}_0(\&g, \&hs, \underline{g.P})$; and (c) change the signature just like 1(c).

```
$\mathrm{SimCap}(\mathbf{bool}\ f, \&\mathbf{sst}\ g, \&\mathbf{st}\ hs, \mathbf{str}\ X)$    $\mathbf{G}_2$ $\boxed{\mathbf{G}_3}$
$\mathbf{if}\ \neg hs.K\ \mathbf{then\ ret}\ (X, \mathrm{mHash}(\&hs, X))$
$(K, A) \leftarrow (hs.K, hs.A);\ N \leftarrow \underline{hs.seq}_{n'}$
$\mathbf{if}\ f\ \mathbf{then}\ Y \leftarrow \mathcal{AE}.\mathrm{Enc}(K, N, A, X)$
$\mathbf{else}\ Y \leftarrow \mathcal{AE}.\mathrm{Dec}(K, N, A, X)$
$\mathbf{if}\ Y = \bot\ \mathbf{then\ ret}\ (\diamond, \texttt{err\_cap})$
$\mathbf{if}\ \neg g.known\ \mathbf{then}\ g.bad_2 \leftarrow 1$
    $\boxed{\mathbf{ret}\ (\diamond, \texttt{err\_cap})}$
$\mathrm{mHash}(\&hs, Y);\ \mathbf{ret}\ (Y, \mathrm{iNonce}(\&hs))$
```

```
$\mathrm{mKeySimDH}^{\mathbf{R}}(\&\mathbf{sst}\ g, \&\mathbf{st}\ hs, \mathbf{elem}_{\mathbb{G}}\ Y)$    $\mathbf{G}_3$ $\boxed{\mathbf{G}_4}$
$\mathbf{dec\ str}\ L', L'';\ Z \in \mathbb{G}$
$\mathbf{if}\ \neg Y\ \mathbf{then\ ret}\ \texttt{err\_dh}$
$Z \twoheadleftarrow \mathcal{P}^{\mathbf{R}}(\&g, \&hs, Y)$
$g.known \leftarrow (Z \wedge (hs.id, hs.L) \in g.\mathcal{U})$
$\mathbf{if}\ g.known\ \mathbf{then}\ (hs.L, L', L'') \twoheadleftarrow \mathbf{R}(u, \underline{Z})$
$\mathbf{else}\ (hs.L, L', L'') \twoheadleftarrow \mathbf{R}(u, \underline{(g.s)Y})$
    $\boxed{X \leftarrow (u, \underline{(g.s)Y})}$
    $\boxed{\mathbf{if}\ \neg g.\pi_X\ \mathbf{then}\ g.\pi_X \twoheadleftarrow (\{0,1\}^h)^3}$
    $\boxed{(hs.L, L', L'') \leftarrow g.\pi_X}$
$hs.K \leftarrow L'[:k];\ hs.seq \leftarrow 0$
$x \leftarrow g.known;\ g.\mathcal{K}_x \leftarrow g.\mathcal{K}_x \cup \{hs.K\}$
$\mathbf{ret}\ \diamond$
```

```
$\mathrm{SimDH}^{\mathbf{DDH}}(\&\mathbf{sst}\ g, \&\mathbf{st}\ hs, \mathbf{elem}_{\mathbb{G}}\ Y)$
$g.\mathcal{R} \leftarrow g.\mathcal{R} \cup \{Y\}$
$\mathbf{for\ each}\ (R, W) \in g.\mathcal{R} \times g.\mathcal{V}\ \mathbf{do}$
    $\mathbf{if}\ \neg g.\rho_R \wedge \mathbf{DDH}(g.P, R, W)\ \mathbf{then}$
        $g.\rho_R \leftarrow W$
$\mathbf{ret}\ g.\rho_Y$
```

```
$\langle \mathcal{S}^{\mathrm{ns}}[\mathcal{P}].\mathrm{Init}\colon \mathbf{DDH}, \mathbf{Q}, \mathbf{R}\rangle(pk)$
$\mathbf{dec\ sst}\ g;\ \underline{g.P} \leftarrow pk;\ \mathbf{ret}\ \underline{g}$

$\langle \mathcal{S}^{\mathrm{ns}}[\mathcal{P}].\mathrm{Op}\colon \mathcal{O}, \mathbf{DDH}, \mathbf{Q}, \mathbf{R}\rangle(\&\sigma, ctx, op, in)$
$\mathbf{dec\ sst}\ g;\ \mathbf{st}\ hs;\ \mathbf{msg}\ req$
$\mathbf{dec\ bool}\ f, r;\ \mathbf{str}\ u, pat, err$
$\underline{g} \leftarrow \sigma;\ \underline{u, f, r, pat} \leftarrow op;\ \underline{hs, in} \leftarrow in$
$\mathbf{for\ each}\ (id, u, v) \in \mathbf{Q}(\,) \setminus g.\mathcal{Q}\ \mathbf{do}\ \underline{Z} \leftarrow v$
    $g.\mathcal{U} \leftarrow g.\mathcal{U} \cup \{(id, u)\}$
    $\mathbf{if}\ Z\ \mathbf{then}\ g.\mathcal{V} \leftarrow g.\mathcal{V} \cup \{Z\}$
$g.\mathcal{Q} \leftarrow g.\mathcal{Q} \cup \mathbf{Q}(\,)$
$\mathbf{if}\ u = \texttt{noise} \wedge hs.id = vr(ctx)\ \mathbf{then}$
    $\mathbf{ret}\ \mathrm{SimOp}[\mathcal{P}]^{\mathbf{DDH},\mathbf{R}}(\&g, ctx, op, in)$
$\mathbf{else\ ret}\ \mathcal{O}(ctx, op, in)$
```

Figure 15: Left: Experiments 3 (top) and 4 (bottom) for the proof of Theorem 7. Right: Procedure SimDH (top) and simulator $\mathcal{S} = \mathcal{S}^{\mathrm{ns}}[\mathcal{P}]$ (bottom) for proof of Theorem 7. Procedure $\mathcal{P}$ has signature $\mathcal{P}(\&\mathbf{sst}\ g, \&\mathbf{st}\ hs, \mathbf{elem}_{\mathbb{G}}\ Y) \mapsto \mathbf{elem}_{\mathbb{G}}\ Z$ (and can be instantiated via SimDH); procedure $\mathrm{SimOp}[\mathcal{P}]$ is as defined in experiment 5; type **sst** is defined in Figure 14; and oracle **DDH** is defined in Figure 4.

3. Define SimWrite from Write as follows: (a) replace execution of $\mathrm{wTok}^{\mathbf{R}}(\&hs, \&out, s, r, \boldsymbol{t}_i)$ with execution of $\mathrm{wSimTok}^{\mathbf{R}}(\&g, \&hs, \&out, r, \boldsymbol{t}_i)$; (b) replace $\mathrm{Cap}_1(\&hs, in)$ with $\mathrm{SimCap}_1(\&g, \&hs, in)$; and (c) change the signature just like 1(c).

4. Define SimRead from Read in kind: (a) replace execution of $\mathrm{rTok}^{\mathbf{R}}(\&hs, in, s, r, \boldsymbol{t}_i)$ with execution of $\mathrm{rSimTok}^{\mathbf{DDH},\mathbf{R}}(\&g, \&hs, in, r, \boldsymbol{t}_i)$; (b) replace $\mathrm{Cap}_0(\&hs, in.P)$ with $\mathrm{SimCap}_0(\&g, \&hs, in.P)$; and (c) change the signature just like 1(c).

5. Define SimOp from Op as follows: (a) replace execution of $\mathrm{Write}^{\mathbf{R}}(\&hs, s, r, pat, in)$ with execution of $\mathrm{SimWrite}^{\mathbf{R}}(\&g, \&hs, r, pat, in)$ immediately preceded by the statement $g.known \leftarrow 1$; (b) replace execution of $\mathrm{Read}^{\mathbf{R}}(\&hs, s, r, pat, req)$ with $\mathrm{SimRead}^{\mathbf{DDH},\mathbf{R}}(\&g, \&hs, r, pat, req)$ immediately preceded by the statement $g.known \leftarrow 1$; (c) remove $s \leftarrow \mathrm{Scal}^{\mathbf{R}}(sk)$ from line 7:11; and (d) change the signature by removing **str** sk and adding $\&$**sst** g as the first argument.

6. Finally, modify the **Op** oracle by replacing $\mathcal{N}^{\mathbf{R}}_{\underline{s}}(ctx, op, in)$ with $\mathrm{SimOp}^{\mathbf{DDH},\mathbf{R}}(\&g, ctx, op, in)$.

We claim that $\mathbf{G}_2(\mathcal{S}, \mathcal{D})$ and $\mathbf{G}_1(\mathcal{S}, \mathcal{D})$ are equivalent. The main change is in the way static DH operations are performed (via mKeySimDH). First note that if SimDH outputs any point at all, then the output is correct; this is made possible by the **DDH** oracle, which allows the procedure to decide if $(\log_G P)(\log_G Y) = \log_G Z$ for the input Y and some $Z \in g.\mathcal{V}$. Thus, if the solution coincides with one of $\mathcal{D}$'s queries, then SimDH will find it in $O(q_R q_I)$ time. Otherwise mKeySimDH will compute the correct solution using $g.s$.

The remaining changes addresses this dependence on $g.s$. First, experiment $\mathbf{G}_3$ is defined from $\mathbf{G}_2$ by modifying SimCap as specified in the top-left panel of Figure 15. The experiments are identical until $g.bad_2$ gets set, after which SimCap halts and outputs $(\diamond, \texttt{err\_cap})$ in $\mathbf{G}_3$. SimCap gets called when encapsulating or decapsulating a payload or static key blob; flag $g.bad_2$ only gets set if encryption/decryption succeeds and

$\neg g.known$ holds. Since variable $g.known$ can only be unset by a read action (mKeySimPSK or mKeySimDH), this event is always triggered by a successful decryption under a key in the set $g.\mathcal{K}_0$.

The set of keys $g.\mathcal{K}_0$ is unknown to $\mathcal{D}$ in the sense that, by definition, each key was derived from an RO query for which at least one of the inputs (i.e., the key state or DH secret), is not incident to any RO query made by $\mathcal{D}$. However, this fact alone does not preclude the possibility of $\mathcal{D}$ learning information about $g.\mathcal{K}_0$, since **Call** or **Op** oracles may leak the handshake state to the adversary. Indeed, upon a successful **Op** query, the adversary gets the symmetric key used to encapsulate/decapsulate the payload, since it is contained in the state; and the **Call** oracle might leak keys if, for example, the game $\mathcal{G}$ admits state-corruption queries (e.g., when modeling forward secrecy). But the (ψ-ro-)regularity of $\mathcal{G}$ ensures that symmetric keys used in the game $\mathcal{G}$ are independent of $g.\mathcal{K}_0$; any keys leaked by the interface are either in $g.\mathcal{K}_1$, or the leakage occured only after $g.bad_2$ got set.

These observations allow us to bound the probability that $\mathbf{G}_3(\mathcal{S},\mathcal{D})$ sets $g.bad_2$ using the INT-CTXT security of $\mathcal{AE}$. Consider the following INT-CTXT adversary. It executes $\mathbf{G}_3(\mathcal{S},\mathcal{D})$, but replaces all encryption or decryption operations incident to some $K \in g.\mathcal{K}_0$ with queries to its own **Enc** or **Dec** oracle. (Note that only **Dec** will be used, since $g.known$ is only unset by read actions.) This is made by possible by virtue of the fact that K is never revealed to $\mathcal{D}$, since the operator will return only an error (err_cap) instead of the handshake state. Such an attack succeeds only if the experiment sets $g.bad_2$ and decryption operation incident to this event was evaluated using **Dec**.

Let $\mathcal{A}_i$ be like the adversary just described, but it simulates actions involving the i-th key entered into $g.\mathcal{K}_0$ using its own oracle. Note that $|g.\mathcal{K}_0| \leq \ell q_I$ and define $\mathcal{A}^{\mathbf{Enc},\mathbf{Dec}}$ as follows: choose $i \twoheadleftarrow [\ell q_I]$, run $\mathcal{A}_i^{\mathbf{Enc},\mathbf{Dec}}$, and halt. Let $\mathbf{H}_i(\mathcal{S},\mathcal{D})$ be an experiment defined just like $\mathbf{G}_2$, except that after i-th key is entered into the set $g.\mathcal{K}_0$, if $g.bad_2$ gets set, then SimCap halts and outputs an error (as in $\mathbf{G}_3$). Then $\mathbf{G}_2$ is the same as $\mathbf{H}_{\ell q_I}$, experiment $\mathbf{G}_3$ is the same as $\mathbf{H}_0$, and

$$\begin{aligned}
\Pr\left[\,\mathbf{G}_2(\mathcal{S},\mathcal{D})\,\right] - \Pr\left[\,\mathbf{G}_3(\mathcal{S},\mathcal{D})\,\right] &= \sum_{i=1}^{\ell q_I} \Pr\left[\,\mathbf{H}_i(\mathcal{S},\mathcal{D})\,\right] - \Pr\left[\,\mathbf{H}_{i-1}(\mathcal{S},\mathcal{D})\,\right] && (55)\\
&\leq \sum_{i=1}^{\ell q_I} \Pr\left[\,\mathbf{H}_i(\mathcal{S},\mathcal{D})\,\textbf{sets}\,g.bad_2\,\right] && (56)\\
&\leq \sum_{i=1}^{\ell q_I} \Pr\left[\,\mathbf{Exp}^{\text{int-ctxt}}_{\mathcal{AE}}(\mathcal{A}_i)\,\right] && (57)\\
&= \ell q_I \mathbf{Adv}^{\text{int-ctxt}}_{\mathcal{AE}}(\mathcal{A})\,, && (58)
\end{aligned}$$

where the last line follows by conditioning on the outcome of $\mathcal{A}$'s sampling i uniformly from $[\ell q_I]$.

Observe that in $\mathbf{G}_3$, the output of SimCap is independent of $g.\mathcal{K}_0$, since if $\neg g.known$ holds, then it outputs $(\diamond,$ err_cap$)$ regardless of the outcome the decryption operation. This allows us to rewrite mKeySimDH so that it is independent of $g.s$. We do so in two steps. First, experiment $\mathbf{G}_4$ is defined from $\mathbf{G}_3$ in the bottom-left panel of Figure 15 by replacing evaluation of $\mathbf{R}$ in case $\neg g.known$ holds (15:14) with a simulation using the shadow state. Specifically, it checks an associative array $g.\pi$ for $X = (u, \underline{(g.s)Y})$; if $g.\pi_X$ is undefined, then sets it to a random element of $(\{0,1\}^h)^3$. It then sets $(hs.L, L', L'') \leftarrow g.\pi_X$. This does not change the distribution of the experiment, however, because $\neg g.known$ implies that π_X is undefined. It follows that

$$\Pr\left[\,\mathbf{Exp}^{\text{gap1}}_{\mathcal{N},\mathcal{G}}(\mathcal{S},\mathcal{D})\,\right] \leq cq_I \mathbf{Adv}^{\text{cr}}_{vr}(\mathcal{C}) + \ell q_I \mathbf{Adv}^{\text{int-ctxt}}_{\mathcal{AE}}(\mathcal{A}) + \Pr\left[\,\mathbf{G}_4(\mathcal{S},\mathcal{D})\,\right]. \quad (59)$$

Finally, define experiment $\mathbf{G}_5$ from $\mathbf{G}_4$ as follows. On line 15:15, replace "$(g.s)Y$" with just "Y". Because n is prime, by Langrange's Theorem, scalar multiplication of an element of $\mathbb{G}$ by $g.s \in \mathbb{Z}_n$ is injective. In particular, it holds that $(g.s)X = (g.s)Y \iff X = Y$ for all $X, Y \in \mathbb{G}$, and so the distribution of the experiment (in particular, the distribution on the key state $hs.L$ and derived values) does not change.

Let SimOp[$\mathcal{P}$] denote SimOp as it is specified in experiment $\mathbf{G}_5$ and where $\mathcal{P}$ is the DH-simulation procedure used to instantiate mKeySimDH. (We write it this way in order to emphasize that, while we use $\mathcal{P} =$ SimDH in the current proof, there may be other instantiations that do not require a **DDH** oracle.) Since SimOp[$\mathcal{P}$] does not depend on the secret key, we may use it as a sub-routine in the specification of our simulator. Let $\mathcal{S} = \mathcal{S}^{\text{ns}}$[SimDH] be as specified in Figure 15. One can easily verify that, when the

```
$\mathbf{Exp}^{\mathrm{indiff}}_{\mathcal{F}_1,\mathcal{F}_0}(\mathcal{T},\mathcal{D})$
    $\mathbf{dec\ str}\ st, \sigma$; $\mathbf{bool}\ b, d$
    $b \twoheadleftarrow \{0,1\}$
    $st \twoheadleftarrow \mathcal{F}_b.\mathrm{Init}(\,)$
    $\sigma \twoheadleftarrow \mathcal{T}.\mathrm{Init}(\,)$
    $d \twoheadleftarrow \mathcal{D}^{\mathbf{Func},\mathbf{Prim}}$
    $\mathbf{ret}\ (b = d)$

$\mathbf{Func}(in)$
    $\mathbf{ret}\ \mathcal{F}_b.\mathrm{Hon}(\&st, in)$

$\mathbf{Prim}(in)$
    $\mathbf{if}\ b = 1\ \mathbf{then}$
        $\mathbf{ret}\ \mathcal{F}_1.\mathrm{Adv}(\&st, in)$
    $\mathbf{ret}\ \mathcal{T}.\mathrm{Op}^{\mathcal{F}_0.\mathrm{Adv}(\&st,\cdot)}(\&\sigma, in)$
```

```
$\mathbf{R}[\mathcal{X},\mathcal{Y}].\mathrm{Init}(\,)$
    $\mathbf{dec\ elem}_{\mathcal{Y}}\ \pi[\,]$; $\mathbf{ret}\ \underline{\pi}$

$\mathbf{R}[\mathcal{X},\mathcal{Y}].\mathrm{Hon}(\&st, in)$
    $\mathbf{dec\ elem}_{\mathcal{X}}\ X$; $\mathbf{elem}_{\mathcal{Y}}\ \pi[\,]$
    $\underline{\pi} \leftarrow st$; $\underline{X} \leftarrow in$
    $\mathbf{if}\ \neg\pi_X\ \mathbf{then}\ \pi_X \twoheadleftarrow \mathcal{Y}$
    $st \leftarrow \underline{\pi}$; $\mathbf{ret}\ \pi_X$

$\mathbf{R}[\mathcal{X},\mathcal{Y}].\mathrm{Adv}(\&st, in)$
    $\mathbf{ret}\ \mathrm{Hon}(\&st, in)$
```

```
$\mathbf{HKDF}[\mathcal{F}, j].\mathrm{Init}(\,)$
    $\mathbf{ret}\ \mathcal{F}.\mathrm{Init}(\,)$

$\mathbf{HKDF}[\mathcal{F}, j].\mathrm{Hon}(\&st, in)$
    $\mathbf{dec\ int}\ j$; $\mathbf{str}\ X, Y, L[\,]$; $\mathbf{tup}\ \boldsymbol{r}$
    $\underline{id, X, Y} \leftarrow in$
    $K \twoheadleftarrow \mathcal{F}.\mathrm{Hon}(\&st, \underline{X, Y})$ // Extract
    $\mathbf{for}\ i \leftarrow 1\ \mathbf{to}\ j\ \mathbf{do}$ // Expand
        $v \leftarrow L_{i-1} \,\|\, id \,\|\, \underline{i}_8$
        $L_i \twoheadleftarrow \mathcal{F}.\mathrm{Hon}(\&st, \underline{K, v})$
        $\boldsymbol{r} \leftarrow \boldsymbol{r} \,\|\, L_i$
    $\mathbf{ret}\ \underline{\boldsymbol{r}}$

$\mathbf{HKDF}[\mathcal{F}, j].\mathrm{Adv}(\&st, in)$
    $\mathbf{ret}\ \mathcal{F}.\mathrm{Adv}(\&st, in)$
```

Figure 16: Left: INDIFF experiment for funcionalities $\mathcal{F}_1$ and $\mathcal{F}_0$, primtive-simulator $\mathcal{T}$, and adversary $\mathcal{D}$. Middle: The random oracle functionality $\mathbf{R}[\mathcal{X},\mathcal{Y}]$ for a function with domain $\mathcal{X}$ and *finite* range $\mathcal{Y}$. Right: The HKDF functionality $\mathbf{HKDF}[\mathcal{F}, j]$ defined in terms of functionality $\mathcal{F}$ (an idealization of HMAC) and integer $0 \le j < 255$.

---

simulator is defined in this way, the output of **Op** in $\mathbf{G}_5(\mathcal{S},\mathcal{D})$ is independent of b. We conclude that $\Pr\big[\,\mathbf{G}_5(\mathcal{S},\mathcal{D})\,\big] = 1/2$. The claimed bound follows.

We conclude by accounting for $\mathcal{C}$'s, $\mathcal{A}$'s, and $\mathcal{S}$'s resources. Both $\mathcal{A}$ and $\mathcal{C}$ are $O(t + q_R q_I)$-time, which accounts for the bookkeeping overhead inherited from experiment $\mathbf{G}_0$. Adversary $\mathcal{A}$ makes no queries to **Enc**, but at most q_I queries to **Dec**. Finally, the simulator is $(O(t/q_I), q_I, \ell)$-resource, $\ell q_R q_I$-**DDH**-bound, and 2-**Q**-bound.

A.7 Sketch of Theorem 8 (GAP2 security of Noise)

The proof of GAP2 security of $(\mathcal{I}_{+\mathcal{N}}, \mathcal{I})$ for $\mathcal{G}$ is closely related to the proof of Theorem 7, so we will only enumerate the differences here. First, experiment $\mathbf{G}_0$ has the following differences:

1. Instead of $(\underline{g.P}, \underline{g.s}) \twoheadleftarrow \mathcal{N}.\mathrm{Gen}(\,)$ we run $(\underline{g.P}, sk) \twoheadleftarrow \mathcal{I}.\mathrm{Gen}(\,)$ and set $g.s \leftarrow \mathcal{I}.\mathrm{Scal}(sk)$. (Recall that $\mathcal{I}.\mathrm{Gen}$ is 0-ro-bound by assumption.)
2. We modify **Final** and **Call** to use $\mathcal{I}_{+\mathcal{N}}.\mathrm{Op}_{sk}$ if $b = 1$ and $\mathcal{I}_{sk}$ otherwise. Both are given $\mathbf{R}_2$ as the RO just as in Theorem 7. Notice that since $\mathcal{I}$ is ψ-ro-regular by assumption, the same condition as before will trigger $g.bad_1$ getting set: if a RO query made by via **Call** coincides with an RO query made via **Op**, then the result will be $\perp$ in $\mathbf{G}_2$.
3. We modify **Op** so that $\mathcal{I}_{+\mathcal{N}}.\mathrm{Op}_{sk}$ is used if $b = 1$ and $\langle \mathcal{S}.\mathrm{Op}_\sigma : \mathcal{I}^{\mathbf{R}}_{sk} \rangle$ is used otherwise.

The remaining steps are the same as in the proof of Theorem 7.

B Indifferentiability of HKDF

In our analysis of the SEC/I security of Noise (Section 6) we modeled the key-derivation function as a random oracle (RO). Noise uses HKDF [38] instantiated with a hash function $\mathcal{H}$. This modeling choice is only valid if the attacker is unable to exploit the underlying structure of HKDF except with negligible proboablity. To formally justify this, we prove that HKDF is indifferentiable from an RO when $\mathcal{H}$ is modeled as an RO.

Definition 11 (Indifferentiable functionalities)**.** We recall the notion of indifferentiable functionalities *for single-stage aversaries* of Ristenpart, Schacham, and Shrimpton [55]. A *functionality* $\mathcal{F}$ is a triple of algorithms (Init, Hon, Adv):

- $\mathrm{Init}(\,) \mapsto \mathbf{str}\ st$. Returns the initial state of the functionality.

- $\mathrm{Hon}, \mathrm{Adv}(\&st, \mathbf{str}\, in) \mapsto \mathbf{str}\, out$. The *honest* and *adversarial* interfaces of the functionality take as input a string in and the current state st and return a string out. Each may update st as a side-effect.

The INDIFF experiment (left panel of Figure 16) is associated with functionalities $\mathcal{F}_1$ and $\mathcal{F}_0$, an adversary $\mathcal{D}$, and an *INDIFF-simulator* $\mathcal{T}$. The INDIFF simulator is a pair of algorithms $(\mathrm{Init}, \mathrm{Op})$: the first takes no inputs and outputs a string σ representing the simulator's initial state; and the second, $\mathrm{Op}(\&\mathbf{str}\,\sigma, \mathbf{str}\, in) \mapsto \mathbf{str}\, out$ processes an input and updates σ. Algorithm Op expects access to an oracle for an adversarial interface. Let $\mathbf{Adv}^{\mathrm{indiff}}_{\mathcal{F}_1,\mathcal{F}_0}(\mathcal{T}, \mathcal{D})$ denote the advantage of $\mathcal{D}$ in differentiating $\mathcal{F}_1$ from $\mathcal{F}_0$ with respect to $\mathcal{T}$. Informally, we will say that $\mathcal{F}_1$ is indifferentiable from $\mathcal{F}_0$ if there exists an efficient INDIFF-simulator $\mathcal{T}$ such that the advantage of any efficient adversary in differentiating $\mathcal{F}_1$ from $\mathcal{F}_0$ with respect to $\mathcal{T}$ is small. We call an INDIFF adversary $(\mathcal{X}_F, \mathcal{X}_P)$*-restricted* if each of its **Func** queries encodes an elment of $\mathcal{X}_F$ and each of its **Prim** queries encodes an element of $\mathcal{X}_P$. We call an INDIFF simulator (t, c)-resource if each of its constituent algorithms are t-time and the operator makes at most c queries to its oracle on any given execution. ♦

HKDF uses HMAC [39] instantiated with a hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^h$. HMAC requires $\mathcal{H}$ be computed by iterating an underlying compression function on fixed-length blocks of data, where the block size is at least h bits. Therfore, in order to instantiate HKDF with a particular hash function, the hash function must be suitable for HMAC. Many widely used hash function are, including SHA2 and BLAKE2 (both specified for use with Noise). Although designed for message authentication, HMAC is used for randomness extraction in a wide variety of settings. As a result, it is common to model HMAC as a random oracle when anaylzing the security of theses schemes.

In the single-stage setting [55] and under appropriate restrictions of the key space, HMAC is known to be indifferentiable from an RO when $\mathcal{H}$ is modeled as an RO [24]. Dodis et al. define (in [24, Section 4.4]) *allowed* key sets for a particular hash function $\mathcal{H}$, and they show that for all allowed key sets $\mathcal{K}$, the function $\mathrm{HMAC}^{\mathcal{H}} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^h$ is indifferentiable from an RO when $\mathcal{H}$ is modeled as an RO. However, to the best of our knowledge, it is not known whether (and under which conditions) HKDF is indifferentiable from an RO. To close this gap, we build on the result of Dodis et al. by showing that HKDF is indifferentiable from an RO when HMAC is modeled as an RO. To prove this, we will have to restrict the key space somewhat more: Dodis et al. point out that for any $k < d - 1$, where d is the block size of $\mathcal{H}$, the set $\{0,1\}^k$ is an allowed key set for $\mathcal{H}$; see [24, Section 4.4]. We will focus our attention on hash function for which $d > h$ (e.g., SHA2 and BLAKE2) and require the salt string used for HKDF to be of length h. That is, we restrict ourselves to the key set $\{0,1\}^h$, which is allowed for the class of hash functions that we consider.

In Figure 16 we express HKDF as a functionality $\mathbf{HKDF}[\mathcal{F}, j]$ defined in terms of a functionality $\mathcal{F}$ and an integer $j \geq 0$. The functionality $\mathcal{F}$ is an idealization of HMAC, which the HKDF's adversarial interface exposes directly (16:26). We realize $\mathcal{F}$ as an *RO functionality*, which is also defined in Figure 16. An RO functionality $\mathbf{R}[\mathcal{X}, \mathcal{Y}]$ is equivalent to modeling some function $H : \mathcal{X} \to \mathcal{Y}$ as an RO in an experiment (see Section 4.2); its initializer declares an associative array used to lazy-compute a random function from $\mathcal{X}$ to $\mathcal{Y}$. Its honest and adversarial interfaces are the same. Note that the RO functionality is only well-defined if $\mathcal{Y}$ is finite.

Fix integers $0 \leq j \leq 255$ and $h \geq 0$ and a function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^h$ suitable for HMAC with block size $d > h$. Let $\mathcal{X}_H = \{0,1\}^h \times \{0,1\}^*$ and let $\mathcal{X}_F = \{0,1\}^* \times \{0,1\}^h \times \{0,1\}^*$. Let $H = \mathbf{R}[\mathcal{X}_H, \{0,1\}^h]$, $F = \mathbf{HKDF}[H, j]$, and $R = \mathbf{R}[\mathcal{X}_F, (\{0,1\}^h)^j]$ as defined in Figure 16.

Theorem 9. *Fix integers $0 \leq q_F, q_P \leq 2^h$. There exists an INDIFF-simulator $\mathcal{T}$ such for every t-time INDIFF-adversary $\mathcal{D}$ making q_F queries to* **Func** *and q_P queries to* **Prim***, it holds that*

$$\mathbf{Adv}^{\mathrm{indiff}}_{F,R}(\mathcal{T}, \mathcal{D}) \leq (j+2)(q_F + q_P)^2/2^h\,,$$

where $\mathcal{D}$ is $(\mathcal{X}_F, \mathcal{X}_H)$-restricted and $\mathcal{T}$ is $(O(t/(q_P+1)), 1)$-resource.

Proof. Without loss of generality, we assume that each of $\mathcal{D}$'s queries to its functionality oracle (i.e., **Func**) are distinct. The proof is by a game-playing argument; refer to Figure 17. We begin with an experiment $\mathbf{G}_0(\mathcal{T}, \mathcal{D})$ that is functionally equivalent to $\mathbf{Exp}^{\mathrm{indiff}}_{F,R}(\mathcal{T}, \mathcal{D})$, but rewritten in order to clarify the argument. The RO functinoalities have been made an explicit part of the experiment: the state of R is maintained by associative array Π, and the state of H is maintained by π. The pseudocode of the HKDF functionality has

```
$\mathbf{G}_0(\mathcal{T},\mathcal{D})$ / $\boxed{\mathbf{G}_1(\mathcal{T},\mathcal{D})}$
dec str $\sigma, \pi[\,]$; tup $\Pi[\,]$
set $\mathcal{Q}$; bool $b, d, bad[\,]$
$\sigma \twoheadleftarrow \mathcal{T}.\mathrm{Init}(\,)$
$b \twoheadleftarrow \{0,1\}$
$d \twoheadleftarrow \mathcal{D}^{\mathbf{Func}_b,\mathbf{Prim}_b}$
ret $(b = d)$

$\mathbf{Func}_0(in)$
str $id, X, Y$
$\underline{id, X, Y} \leftarrow in$
$\boldsymbol{q} \twoheadleftarrow \mathbf{R}(id, X, Y)$
ret $\boldsymbol{q}$

$\mathbf{Prim}_0(in)$
ret $\mathcal{T}.\mathrm{Op}^{\mathbf{Func}_0}(\&\sigma, in)$

$\mathbf{R}(id, X, Y)$
if $\neg\Pi_{id,X,Y}$ then $\Pi_{id,X,Y} \twoheadleftarrow (\{0,1\}^h)^j$
ret $\Pi_{id,X,Y}$

$\mathbf{H}(U, V)$
if $\neg\pi_{U,V}$ then $\pi_{U,V} \twoheadleftarrow \{0,1\}^h$
ret $\pi_{U,V}$
```

```
$\mathbf{Func}_1(in)$
dec int $j$; str $X, Y, L[\,]$; tup $\boldsymbol{r}$
$\underline{id, X, Y} \leftarrow in$
$K \twoheadleftarrow \mathbf{H}(X, Y)$ // Extract
if $\neg\pi_{X,Y}$ then $K \twoheadleftarrow \{0,1\}^h$
   if $K \in \mathcal{Q}$ then $bad_0 \leftarrow 0$
   $\pi_{X,Y} \leftarrow K$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{K\}$
$K \leftarrow \pi_{X,Y}$ // Extract
for $i \leftarrow 1$ to $j$ do // Expand
   $v \leftarrow L_{i-1} \,\|\, id \,\|\, \underline{i}_8$
   $L_i \twoheadleftarrow \mathbf{H}(K, v)$
   $\boldsymbol{r} \leftarrow \boldsymbol{r} \,\|\, L_i$
ret $\underline{\boldsymbol{r}}$

$\mathbf{Prim}_1(in)$
dec str $U, V$
$\underline{U, V} \leftarrow in$; ret $\mathbf{H}(U, V)$
if $\neg\pi_{U,V}$ then $W \twoheadleftarrow \{0,1\}^h$
   if $W \in \mathcal{Q}$ then $bad_{256} \leftarrow 0$
   $\pi_{U,V} \leftarrow W$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{W\}$
ret $\pi_{U,V}$
```

```
$\mathbf{Func}_1(in)$                                    $\mathbf{G}_1$ $\boxed{\mathbf{G}_2}$
dec int $j$; str $X, Y, L[\,]$; tup $\boldsymbol{r}$
$\underline{id, X, Y} \leftarrow in$
if $\neg\pi_{X,Y}$ then $K \twoheadleftarrow \{0,1\}^h$
   if $K \in \mathcal{Q}$ then $bad_0 \leftarrow 0$; $\boxed{K \twoheadleftarrow \{0,1\}^h \setminus \mathcal{Q}}$
   $\pi_{X,Y} \leftarrow K$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{K\}$
$K \leftarrow \pi_{X,Y}$ // Extract
for $i \leftarrow 1$ to $j$ do // Expand
   $v \leftarrow L_{i-1} \,\|\, id \,\|\, \underline{i}_8$
   $L_i \twoheadleftarrow \mathbf{H}(K, v)$
   $\boldsymbol{r} \leftarrow \boldsymbol{r} \,\|\, L_i$
ret $\underline{\boldsymbol{r}}$
```

```
$\mathbf{Func}_1(in)$                                    $\mathbf{G}_2$ $\boxed{\mathbf{G}_3}$
dec int $j$; str $X, Y, L[\,]$; tup $\boldsymbol{r}$
$\underline{id, X, Y} \leftarrow in$
if $\neg\pi_{X,Y}$ then $K \twoheadleftarrow \{0,1\}^h$
   if $K \in \mathcal{Q}$ then $bad_0 \leftarrow 0$; $K \twoheadleftarrow \{0,1\}^h \setminus \mathcal{Q}$
   $\pi_{X,Y} \leftarrow K$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{K\}$
$K \leftarrow \pi_{X,Y}$ // Extract
$\boldsymbol{q} \twoheadleftarrow \mathbf{R}(id, X, Y)$
for $i \leftarrow 1$ to $j$ do // Expand
   $v \leftarrow L_{i-1} \,\|\, id \,\|\, \underline{i}_8$
   $L_i \leftarrow \boldsymbol{q}_i$
   if $\pi_{K,v}$ then $bad_i \leftarrow 1$; $L_i \leftarrow \pi_{K,v}$
   $\pi_{K,v} \leftarrow L_i$
   $L_i \twoheadleftarrow \mathbf{H}(K, v)$
   $\boldsymbol{r} \leftarrow \boldsymbol{r} \,\|\, L_i$
ret $\underline{\boldsymbol{r}}$
```

Figure 17: Experiments 0 — 3 for the proof of Theorem 9.

been explicitly transcribed by oracles $\mathbf{Func}_1$ and $\mathbf{Prim}_1$, which are given to $\mathcal{D}$ in case $b = 1$; calls to H have been replaced by calls to an oracle $\mathbf{H}$, which populates π. Finally, in case $b = 0$, adverasry $\mathcal{D}$ is given oracles $\mathbf{Func}_0$ and $\mathbf{Prim}_0$: the former interprets the input *in* precisely as R.Hon does, but makes calls to an oracle $\mathbf{R}$ for populating Π; and the latter executes the simulator with access to $\mathbf{Func}_0$.

Next, experiment $\mathbf{G}_1$ replaces the call to $\mathbf{H}$ by $\mathbf{Prim}_1$ and the call to $\mathbf{H}$ by $\mathbf{Func}_1$ made during the extraction phase (i.e., the first call) with explicit, functionally-equivalent psuedocode. For each (X, Y), oracle $\mathbf{Func}_1$ samples a fresh K from $\{0,1\}^h$ and adds it to a set $\mathcal{Q}$; if ever the sampled K already appears in $\mathcal{Q}$, then the experiment sets a flag bad_0. Similarly, for each distinct (U, V), oracle $\mathbf{Prim}_1$ samples a fresh W and adds it to $\mathcal{Q}$; if ever W is already in $\mathcal{Q}$, then the experiment sets a flag bad_{256}. Neither change affects the outcome of the game, and so for every $\mathcal{T}$ and $\mathcal{D}$ it holds that

$$\mathbf{Adv}^{\text{indiff}}_{F,H}(\mathcal{T}, \mathcal{D}) = 2 \Pr\left[\, \mathbf{G}_1(\mathcal{T}, \mathcal{D}) \,\right] - 1 \,. \tag{60}$$

Next, experiment $\mathbf{G}_2$ is identical to $\mathbf{G}_1$ until bad_0 gets set. In the new experiment, if $K \in \mathcal{Q}$, then K is *re-sampled* from $\{0,1\}^h \setminus \mathcal{Q}$. This ensures a one-to-one correspondence between pairs (X, Y) and derivation keys K. The probability that the i-th query to $\mathbf{Func}_1$ sets bad_0 is at most $(q_P + i)/2^h$; summing over all q_F queries, we have that

$$\Pr\left[\, \mathbf{G}_1(\mathcal{T}, \mathcal{D})\, \textbf{sets}\ bad_0 \,\right] \;\le\; \frac{q_F q_P}{2^h} + \sum_{i=1}^{q_F} \frac{i}{2^h} \tag{61}$$

$$= \; \frac{q_F q_P}{2^h} + \frac{(q_F)(q_F - 1)}{2^{h+1}} \,. \tag{62}$$

Next, in $\mathbf{G}_3$ we modify $\mathbf{Func}_1$ by replacing calls to $\mathbf{H}$ made during the expansion phase (17:57) with explicit, functionally-equivalent pseudocode. Instead of populating π with extracted keys using $\mathbf{H}$, we use a call to $\mathbf{R}$ (17:51): for each $i \in [j]$ such that $\pi[K, L_{i-1} \,\|\, id \,\|\, \underline{i}_8]$ is undefined, we let let $L_i = \boldsymbol{q}_i$ and program π appropriately. Since there is a one-to-one correspondince between (X, Y)'s and K's, and $\mathbf{R}$ is only used if $b = 0$, these experiments are identically distributed.

Now refer to experiment $\mathbf{G}_4$ in Figure 18. It is identical to $\mathbf{G}_3$ until bad_i gets set for some $i \in [j]$. Because each of $\mathcal{D}$'s functionality queries are distinct by assumption, this occurs only if $\mathcal{D}$ previously asked $(K, L_{i-1} \,\|\, id \,\|\, \underline{i}_8)$ of its primitive oracle for some $K \in \mathcal{Q}$ and $i \in [j]$. Summing over all of the adversary's primitive-oracle queries, we have that

$$\sum_{i=1}^{j} \Pr\left[\, \mathbf{G}_3(\mathcal{T}, \mathcal{D})\, \textbf{sets}\ bad_j \,\right] \le \frac{j q_F q_P}{2^h} \,. \tag{63}$$

Finally, experiment $\mathbf{G}_5$ (right panel of Figure 18) is functionally equivalent to $\mathbf{G}_4$, but the specification of $\mathbf{Func}_1$ has been simplified by removing the bookkeeping code that is no longer needed. Experiments $\mathbf{G}_6$ and $\mathbf{G}_5$ are equivalent until bad_{256} gets set. In the new experiment, if $W \in \mathcal{Q}$, then W is re-sampled from $\{0,1\}^h \setminus \mathcal{Q}$. As a result, the output of $\mathbf{Prim}_1(U, V)$ is distinct for each distinct (U, V). Indeed, every point in π not programmed by $\mathbf{Func}_1$ (line 18:43) is now guaranteed to be distinct. Moreover,

$$\Pr\left[\, \mathbf{G}_5(\mathcal{T}, \mathcal{D})\, \textbf{sets}\ bad_{256} \,\right] \;\le\; \frac{q_F q_P}{2^h} + \frac{(q_P)(q_P - 1)}{2^{h+1}} \,. \tag{64}$$

To complete the proof, we specify an INDIFF simulator $\mathcal{T}$ such that oracles $(\mathbf{Func}_1, \mathbf{Prim}_1)$ and $(\mathbf{Func}_0, \mathbf{Prim}_0)$ are functionally equivalent. Refer to $\mathcal{T}$ defined in the bottom-left panel of FIgure 18. The simulator maintains a set $\mathcal{Q}$, initially empty. It simulates extraction (18:30–33) on (X, Y) by sampling $K \twoheadleftarrow \{0,1\}^h \setminus \mathcal{Q}$ and setting $T_{X,Y} \leftarrow K$. It also constructs a reverse-lookup table S by setting $S_K \leftarrow (X, Y)$. To ensure a one-to-one correspondence between (X, Y)'s and K's, it adds K to the set $\mathcal{Q}$. Simulating expansion (18:20–29) is more involved. On input of (K, v), it checks to see if S_K is defined. If so, it sets $(X, Y) \leftarrow S_K$ and attempts to parse v as a string $L_{i-1} \,\|\, id \,\|\, \underline{i}_8$ such that $|L_{i-1}| = h$ if $j > 1$ and $|L_{i-1}| = 0$ otherwise. It then queries $\boldsymbol{q} \twoheadleftarrow \mathbf{Func}_1(id, X, Y)$. Now, if $\boldsymbol{q}_{i-1} = L_{i-1}$, $0 < i \le j$, and $K \in \mathcal{Q}$, then the simulator treats the query as an expansion and returns $\boldsymbol{q}_i$; otherwise it treats it as an extraction, as above. This is precisely the behavior of the $\mathbf{Prim}_1$ oracle in $\mathbf{G}_6$, whose output is either those programmed during

```
$\mathbf{Func}_1(in)$                                   $\mathbf{G}_3$  $\mathbf{G}_4$
dec int $j$; str $X, Y, L[\,]$; tup $\boldsymbol{r}$
$\underline{id, X, Y} \leftarrow in$
if $\neg\pi_{X,Y}$ then $K \twoheadleftarrow \{0,1\}^h$
    if $K \in \mathcal{Q}$ then $bad_0 \leftarrow 0$; $K \twoheadleftarrow \{0,1\}^h \setminus \mathcal{Q}$
    $\pi_{X,Y} \leftarrow K$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{K\}$
$K \leftarrow \pi_{X,Y}$ // Extract
$\boldsymbol{q} \twoheadleftarrow \mathbf{R}(id, X, Y)$
for $i \leftarrow 1$ to $j$ do // Expand
    $v \leftarrow L_{i-1} \,\|\, id \,\|\, \underline{i}_8$
    $L_i \leftarrow \boldsymbol{q}_i$
    if $\pi_{K,v}$ then $bad_i \leftarrow 1$; $L_i \leftarrow \pi_{K,v}$
    $\pi_{K,v} \leftarrow L_i$
    $\boldsymbol{r} \leftarrow \boldsymbol{r} \,\|\, L_i$
ret $\underline{\boldsymbol{r}}$
```

```
$\mathcal{T}$.Init( )
dec str $T[\,]$; tup $S[\,]$; set $\mathcal{Q}$
ret $\underline{\mathcal{Q}, S, T}$
```

```
$\mathcal{T}.\mathrm{Op}^{\mathbf{Func}_0}(\&\sigma, in)$
dec str $id, X, Y, U, V, L[\,], T[\,]$
dec tup $\boldsymbol{q}, S[\,]$; int $i$; set $\mathcal{Q}$
$\underline{\mathcal{Q}, S, T} \leftarrow \sigma$; $\underline{U, V} \leftarrow in$
// Try to expand first.
$K \leftarrow U$; $(X, Y) \leftarrow S_K$
$\underline{i}_8 \leftarrow V[|V| - 7{:}]$ // Most significant byte
if $i > 1$ then $L_{i-1} \leftarrow V[{:}h]$; $id \leftarrow V[h{:}|V| - 8]$
if $i = 1$ then $L_{i-1} \leftarrow \varepsilon$; $id \leftarrow V[{:}|V| - 8]$
$\boldsymbol{q} \twoheadleftarrow \mathbf{Func}_0(\underline{id, X, Y})$
if $\boldsymbol{q}_{i-1} = L_{i-1}$ // Note that $\boldsymbol{q}_0 = \diamond$ by def.
      $\wedge\, K \in \mathcal{Q}$
      $\wedge\, 0 < i \leq j$ then
    ret $\boldsymbol{q}_i$
else // If can't expand, then extract.
    if $\neg T_{U,V}$ then $T_{U,V} \twoheadleftarrow \{0,1\}^h \setminus \mathcal{Q}$
    $S[T_{U,V}] \leftarrow (U, V)$; $Q \leftarrow Q \cup \{K\}$
    $\sigma \leftarrow \underline{\mathcal{Q}, S, T}$; ret $T_{U,V}$
```

```
$\mathbf{Func}_1(in)$                                   $\mathbf{G}_5$  $\boxed{\mathbf{G}_6}$
dec int $j$; str $X, Y, L[\,]$
$\underline{id, X, Y} \leftarrow in$
if $\neg\pi_{X,Y}$ then $K \twoheadleftarrow \{0,1\}^h \setminus \mathcal{Q}$
    $\pi_{X,Y} \leftarrow K$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{K\}$
$K \leftarrow \pi_{X,Y}$ // Extract
$\boldsymbol{q} \twoheadleftarrow \mathbf{R}(id, X, Y)$
for $i \leftarrow 1$ to $j$ do // Expand
    $L_i \leftarrow \boldsymbol{q}_i$
    $v \leftarrow L_{i-1} \,\|\, id \,\|\, \underline{i}_8$
    $\pi_{K,v} \leftarrow L_i$
ret $\boldsymbol{q}$
```

```
$\mathbf{Prim}_1(in)$
dec str $U, V$
$\underline{U, V} \leftarrow in$
if $\neg\pi_{U,V}$ then $W \twoheadleftarrow \{0,1\}^h$
    if $W \in \mathcal{Q}$ then $bad_{256} \leftarrow 0$
        $\boxed{W \twoheadleftarrow \{0,1\}^h \setminus \mathcal{Q}}$
    $\pi_{U,V} \leftarrow W$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{W\}$
ret $\pi_{U,V}$
```

```
$\mathbf{Func}_0(in)$
str $id, X, Y$
$\underline{id, X, Y} \leftarrow in$
$\boldsymbol{q} \twoheadleftarrow \mathbf{R}(id, X, Y)$
ret $\boldsymbol{q}$
```

```
$\mathbf{Prim}_0(in)$
ret $\mathcal{T}.\mathrm{Op}^{\mathbf{Func}_0}(\&\sigma, in)$
```

Figure 18: Experiments 4 — 6 and INDIFF simulator $\mathcal{T}$ for the proof of Theorem 9.

expansion of $\mathbf{Func}_1$ (18:43), or else for each distinct (U, V), the output $\mathbf{Prim}_1(U, V)$ is sampled without replacement from the set of h-bit strings. It follows that $\Pr\big[\,\mathbf{G}_6(\mathcal{T}, \mathcal{D})\,\big] = 1/2$. Summarizing,

$$
\begin{aligned}
\mathbf{Adv}^{\text{indiff}}_{F,R}(\mathcal{T}, \mathcal{D}) &\leq 2\left(\frac{q_F q_P}{2^h} + \frac{(q_F)(q_F-1)}{2^{h+1}} + \frac{j q_F q_P}{2^h} + \frac{q_F q_P}{2^h} + \frac{(q_P)(q_P-1)}{2^{h+1}} + \frac{1}{2}\right) - 1 && (65)\\
&= \frac{2q_F q_P}{2^h} + \frac{2(q_F)(q_F-1)}{2^{h+1}} + \frac{2j q_F q_P}{2^h} + \frac{2q_F q_P}{2^h} + \frac{2(q_P)(q_P-1)}{2^{h+1}} && (66)\\
&\leq \frac{2q_F q_P}{2^h} + \frac{(q_F)^2}{2^h} + \frac{2j q_F q_P}{2^h} + \frac{2q_F q_P}{2^h} + \frac{(q_P)^2}{2^h} && (67)\\
&= \frac{(q_F)^2}{2^h} + \frac{2(j+2) q_F q_P}{2^h} + \frac{(q_P)^2}{2^h} && (68)\\
&\leq \frac{(j+2)(q_F+q_P)^2}{2^h}\,. && (69)
\end{aligned}
$$

□

Discussion. In order to use the indifferentiability of HKDF from an RO in the analysis of a scheme that uses HKDF—in particular, for Theorem 7 or Theorem 8—it is necessary that the scheme (1) use a hash function that is suitable for HMAC and with block size strictly greater than the output size h, and (2) ensure that the length of the salt is exactly h bits long. This is why our interface for Noise enforces the length of the salt (7:11). These restrictions are mild and are met by many applications of HKDF, including Noise [52]. Nevertheless, we must stress their importance for mitigating attacks against HMAC pointed out by Dodis et al. [24].