Access Control Encryption for General Policies from Standard Assumptions*

Sam Kim and David J. Wu

Stanford University

Abstract. Functional encryption enables fine-grained access to encrypted data. In many scenarios, however, it is important to control not only what users are allowed to *read* (as provided by traditional functional encryption), but also what users are allowed to *send*. Recently, Damgård et al. (TCC 2016) introduced a new cryptographic framework called *access control encryption* (ACE) for restricting information flow within a system in terms of both what users can read as well as what users can write. While a number of access control encryption schemes exist, they either rely on strong assumptions such as indistinguishability obfuscation or are restricted to simple families of access control policies.

In this work, we give the first ACE scheme for *arbitrary* policies from standard assumptions. Our construction is generic and can be built from the combination of a digital signature scheme, a predicate encryption scheme, and a (single-key) functional encryption scheme that supports randomized functionalities. All of these primitives can be instantiated from standard assumptions in the plain model and therefore, we obtain the first ACE scheme capable of supporting general policies from standard assumptions. One possible instantiation of our construction relies upon standard number-theoretic assumptions (namely, the DDH and RSA assumptions) and standard lattice assumptions (namely, LWE). Finally, we conclude by introducing several extensions to the ACE framework to support dynamic and more fine-grained access control policies.

1 Introduction

In the last ten years, functional encryption [16,44] has emerged as a powerful tool for enforcing fine-grained access to encrypted data. But in many real-world scenarios, system administrators need to restrict not only what users are allowed to *read*, but also, what users are allowed to *send*—for example, users with top-secret security clearance in a system should not be able to make sensitive information publicly available. Recently, Damgård, Haagh, and Orlandi [23] introduced the notion of access control encryption (ACE) to enable cryptographic control of the information flow within a system

Access control encryption. An access control encryption scheme [23] provides a cryptographic mechanism for restricting information flow in a system, both in

*The full version of this paper is available at https://eprint.iacr.org/2017/467.pdf.

terms of what parties can read, as well as in terms of what parties can write. Of course, cryptography alone is insufficient here since a malicious sender can always broadcast sensitive messages in the clear. To address this, Damgård et al. [23] introduce an additional party called the *sanitizer*. All communication between senders and receivers is routed through the sanitizer, which performs some processing on the message before broadcasting it to the receivers. The goal in access control encryption is to simplify the operation of the sanitizer so that its function can be outsourced to a party that is only trusted to execute correctly (in particular, the sanitizer does not need to know either the identity of the sender or receiver of each message, nor the security policy being enforced).

Concretely, an ACE scheme is defined with respect to a set of senders $\mathcal{S}$, a set of receivers $\mathcal{R}$, and an access control policy $\pi\colon \mathcal{S} \times \mathcal{R} \to \{0,1\}$, where $\pi(S,R) = 1$ if a receiver $R \in \mathcal{R}$ is allowed to read messages from sender $S \in \mathcal{S}$ (and vice versa). Each sender S has an encryption key ek_S and each receiver R has a decryption key dk_R. To send a message m, the sender first encrypts $\mathsf{ct} \leftarrow \mathsf{ACE.Encrypt}(\mathsf{ek}_S, m)$ and sends ct to the sanitizer. The sanitizer performs some simple processing on ct to obtain a new ciphertext ct', which it broadcasts to all of the receivers. The correctness requirement of an ACE scheme is that if $\pi(S,R) = 1$, then $\mathsf{ACE.Decrypt}(\mathsf{dk}_R, \mathsf{ct}') = m$. Critically, the sanitizer does not know the identities of the sender or receiver, nor does it know the policy π.

The security requirements of an ACE scheme mirror those in the Bell-LaPadula [7] security model. In particular, the *no-read rule* requires that any set of unauthorized receivers $\{R_j\}$ (even in collusion with the sanitizer) cannot learn any information from a sender S if $\pi(S, R_j) = 0$ for all j. The *no-write rule* says that no set of (possibly malicious) senders $\{S_i\}$ can transfer any information to any set of (possibly malicious) receivers $\{R_j\}$ if $\pi(S_i, R_j) = 0$ for all i, j.

Existing constructions of ACE. Damgård et al. [23] gave two constructions of ACE capable of supporting arbitrary policies $\pi\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ (here, the senders and receivers are represented as n-bit identities). Their first construction takes a brute-force approach and is based on standard number-theoretic assumptions such as the decisional Diffie-Hellman assumption (DDH) or the decisional composite residuosity assumption (DCR). The limitation, however, is that ciphertexts in their construction grow *exponentially* in n, thus rendering the scheme inefficient when the set of identities is large. Their second construction is more efficient (the ciphertext size is polylogarithmic in n), but relies on the full power of indistinguishability obfuscation ($i\mathcal{O}$) [6,29].

Subsequently, Fuchsbauer et al. [28] showed how to construct access control encryption for restricted classes of predicates (i.e., equality, comparisons, and interval membership) from standard assumptions on bilinear maps—namely, the symmetric external Diffie-Hellman assumption (SXDH). While their constructions are asymptotically efficient (their ciphertexts are linear in n), the functionalities they can handle are specialized to a restricted class of predicates.

Recently, Tan et al. [57] showed how to instantiate the Damgård et al. brute-force construction using the learning with errors (LWE) assumption. Since their

Construction	Predicate	Ciphertext Size	Assumption
Damgård et al. [23, §3]	arbitrary	$O(2^n)$	DDH or DCR
Damgård et al. [23, §4]	arbitrary	$\mathsf{poly}(n)$	$i\mathcal{O}$
Fuchsbauer et al. [28]	restricted	$\mathsf{poly}(n)$	SXDH
Tan et al. [57]	arbitrary	$O(2^n)$	LWE
This work	arbitrary	$\mathsf{poly}(n)$	DDH, RSA, and LWE

Table 1: Concrete comparison of the ACE construction in this work with previous ACE constructions [23,28,57] for predicates $\pi\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. For the predicate class, we write "arbitrary" if the scheme can support arbitrary access control policies and "restricted" if it can only handle a small set of access control policies (e.g., equality, comparisons, interval testing).

construction follows the Damgård et al. approach, ciphertexts in their construction also grown exponentially in n.

A natural question is whether it is possible to construct an asymptotically-efficient ACE scheme for *arbitrary* functionalities without relying on powerful assumptions like indistinguishability obfuscation. In this work, we show that under standard assumptions (for instance, the DDH, RSA, and LWE assumptions suffice), we obtain an *asymptotically-efficient* ACE scheme for *general* policies.

1.1 Our Contributions

Our main contribution in this work is a new construction of access control encryption that is asymptotically efficient, supports arbitrary policies, and relies only on simple, well-studied assumptions. All previous constructions of ACE were either inefficient, restricted to simple policies, or relied on indistinguishability obfuscation. We refer to Table 1 for a comparison with the state-of-the-art.

In this work, we give a *generic construction* of access control encryption from three main ingredients: a digital signature scheme, a general-purpose predicate encryption scheme [32], and a (single-key) functional encryption scheme that supports randomized functionalities [33,1]. We give a high-level overview of our construction here and provide the formal description in Section 3. In Section 3.1, we show how to instantiate the underlying primitives to obtain an ACE scheme from standard assumptions. Our work thus resolves the main open question posed by Damgård et al. [23] on constructing asymptotically-efficient ACE schemes for arbitrary functionalities from standard assumptions.

Starting point: predicate encryption. First, we review the syntax of a predicate encryption scheme. In a predicate encryption scheme [17,56,36], ciphertexts are associated with a message m in addition to a set of attributes x, and secret keys are associated with functions f. Decrypting a ciphertext associated with

an attribute-message pair (x, m) using a secret key for a function f outputs m if and only if $f(x) = 1$. Moreover, ciphertexts in a predicate encryption scheme hide *both* the attribute x as well as the message m from all parties that are not able to decrypt the ciphertext.[1] Not surprisingly, a predicate encryption scheme that supports general policies can be used to obtain a primitive that resembles an access control encryption scheme. Each sender's encryption key is just the public key for the predicate encryption scheme. To encrypt a message m, the sender encrypts m with its identity as the attribute (i.e., an n-bit string). The sanitizer would simply forward the ciphertext along. The decryption key for a receiver R is a predicate encryption key that implements the policy $\pi(\cdot, R)$. Of course, because the sanitizer simply broadcasts the sender's message to the receivers, this basic scheme does not satisfy the no-write rule. A malicious sender can simply broadcast the message in the clear.

Sanitizing the ciphertext. To provide security against malicious senders, the sanitizer must perform some kind of re-randomization of the sender's ciphertexts. Damgård et al. [23] achieve this by introducing the notion of "sanitizable functional encryption," which is a functional encryption scheme that supports re-randomization of ciphertexts. However, constructing sanitizable functional encryption seems to require indistinguishability obfuscation. In this work, we take a different strategy similar in spirit to proxy re-encryption [4]. Specifically, we view the sanitizer as implementing a "proxy" that takes as input a sender's ciphertext (under some encryption scheme) and then *re-encrypts* that ciphertext under the predicate encryption scheme (with the attribute set to the sender's identity). The guarantee we seek is that the output of the sanitizer is either $\perp$ (if the input ciphertext is invalid) or a *fresh* encryption of the sender's message under the predicate encryption scheme. With this guarantee, the no-read and no-write properties reduce to the security of the predicate encryption scheme.

The problem of building ACE thus reduces to constructing a suitable proxy re-encryption scheme. Here, we rely on a single-key functional encryption for randomized functionalities [33,1]. In a standard functional encryption [16,44] (FE) scheme, secret keys are associated with functions f and ciphertexts are associated with messages m. The combination of a decryption key for a function f and a ciphertext for a message m should together reveal $f(m)$ and nothing more. Recently, Alwen et al. [2] and Goyal et al. [33] extended the notion of functional encryption to also consider issuing keys for randomized functionalities.

A (general-purpose) FE scheme that supports randomized functionalities immediately gives a way of implementing the proxy re-encryption functionality for the sanitizer. First, to encrypt a message m, sender S encrypts the pair (S, m) under the FE scheme. The sanitizer is given a functional key for the re-encryption function that takes as input a pair (S, m) and outputs a predicate encryption of m with attribute S. The receivers' behavior is unchanged. By appealing to the correctness and security of the FE scheme, the sanitizer's output is distributed

[1] This is in contrast to the weaker notion of attribute-based encryption [52,34,12] where the attribute is public.

like a fresh predicate encryption ciphertext.[2] Importantly for our construction, the FE scheme only needs to support issuing a *single* decryption key (for the sanitizer). This means that it is possible to instantiate the FE scheme from standard assumptions (i.e., by applying the transformation in [1] to standard FE constructions such as [51,31,30]). Our construction is conceptually similar to the approach in [23] based on sanitizable FE. In Remark 3.5, we compare our approach to the one in [23] and highlight the key differences that allow us to avoid the need for indistinguishability obfuscation (as seemingly needed for sanitizable FE), and thus, base our construction on simple assumptions.

Signatures for policy enforcement. The remaining problem with the above construction is that the sender has the freedom to choose the identity S at encryption time. Thus, a malicious sender could choose an arbitrary identity and trivially break the no-write security property. We address this by requiring the sender "prove" its identity to the sanitizer when submitting its ciphertext (but without revealing its identity to the sanitizer in the process). This can be done using a standard technique described in [23] (and also applied in several other contexts [10,19]) by giving each sender S a signature σ_S on its identity (included as part of the sender's encryption key). Then, to encrypt a message m, the sender would construct an FE ciphertext for the tuple (S, σ_S, m) containing its identity, the signature on its identity, and the message. The sanitizer's FE key then implements a re-encryption function that first checks the validity of the signature on the identity before outputting a fresh predicate encryption of the message m (still with attribute S). Thus, a malicious sender is only able to produce valid ciphertexts associated with identities for which it possesses a valid signature. With this modification, we can show that the resulting construction is a secure ACE scheme (Theorems 3.3 and 3.4).

Instantiating our construction. Our construction above gives a generic construction of ACE from digital signatures, predicate encryption, and a single-key general-purpose functional encryption scheme for randomized functionalities. In Section 3.1, we show that all of the requisite building blocks of our generic construction can be instantiated from standard assumptions. In particular, security can be reduced to the decisional Diffie-Hellman (DDH) assumption [14], the RSA assumption [49], and the learning with errors (LWE) assumption [48]. This yields the first ACE scheme that supports general policies from standard assumptions.

Extending ACE. In Section 4, we describe several extensions to the notion of ACE that naturally follow from our generic construction. We primarily view

[2] In the actual construction, satisfying the no-write property requires the stronger property that decrypting a *maliciously-generated* ciphertext, say, from a corrupt sender, also yields a fresh ciphertext under the predicate encryption scheme. This is the notion of security against malicious encrypters first considered in [33] and subsequently extended in [1]. The work of [1] shows how to obtain functional encryption for randomized functionalities with security against malicious encrypters from any functional encryption scheme supporting deterministic functionalities in conjunction with standard number-theoretic assumptions.

these extensions as ways of augmenting the schema of access control encryption to provide increased flexibility or to support additional functionalities, and not as qualitatively new properties specific to our particular construction. Indeed, the $i\mathcal{O}$-based construction of Damgård et al. [23] can also be extended to achieve these properties. Our primary contribution is showing that we can achieve these stronger properties without relying on $i\mathcal{O}$. We briefly summarize our main extensions:

- **Dynamic policies:** In the standard notion of ACE [23], the access control policy is specified at the time of system setup. In realistic scenarios, senders and receivers may need to be added to the system, and moreover, access control policies can evolve over time. In Section 4.1, we show that our ACE construction allows us to associate an access control policy specific to each receiver's decryption key. Thus, each receiver's policy can be determined at the time of receiver key generation rather than system setup, which enables a dynamic specification of access control policies.
- **Fine-grained sender policies:** The standard notion of ACE only considers policies expressible as a function of the sender's and receiver's identities. In many scenarios, we may want to impose additional restrictions on the types of messages that a sender could send. For instance, a sender could be allowed to send messages to any receiver with top-secret security clearance, but we want to ensure that all of the messages they send contains a signature from both the sender as well as their supervisor (who would certify the contents of the message). In Section 4.2, we show that a straightforward extension of our construction allows us to additionally enforce policies on the types of messages a user is allowed to send. We also introduce a new security notion for ACE that captures the property that a sender should only be allowed to send messages that conform to their encryption policy.
- **Beyond all-or-nothing decryption:** In a standard ACE scheme, decryption is "all-or-nothing:" receivers who are authorized to decrypt a particular ciphertext are able to do so and learn the underlying message, while receivers who are not authorized to decrypt learn nothing about the message. Just as functional encryption extends beyond all-or-nothing encryption by enabling decrypters to learn partial information about an encrypted message, we can consider a functional encryption analog of access control encryption where receivers are allowed to learn only partial information about messages in accordance with the precise access control policies of the underlying scheme. As a concrete example, an analyst with secret security clearance might only be authorized to learn the metadata of a particular encrypted communication, while an analyst with top-secret security clearance might be authorized to recover the complete contents of the communication. In a "functional ACE" scheme, decryption keys are associated with functions and the decryption algorithm computes a function on the underlying message. In the full version [38], we show how our ACE scheme can be easily extended to obtain a functional ACE scheme.

Concurrent work. Concurrent to this work, Badertscher et al. [5] introduced several strengthened security notions for access control encryption such as security

against chosen ciphertext attacks (CCA-security). They then show how to extend the ACE scheme (for restricted policies) in [28] to achieve their new security notions. In contrast, our focus in this work is constructing an ACE scheme (under the original security notions from [23]) for arbitrary policies from standard assumptions.

Open problems. We leave as an open problem the construction of an ACE scheme (for general policies) where the sanitizer key can be public. This is the case for the ACE construction for restricted policies in [28], but not the case for our construction or the $i\mathcal{O}$-based construction in [23]. Another open problem is constructing an ACE scheme that provides full sender anonymity (see Remark 2.6 for more details). Notably, this is possible from $i\mathcal{O}$ [23], but seems non-trivial from standard assumptions.

1.2 Additional Related Work

Information flow control is a widely studied topic in computer security (see, for instance [7,24,25,53,54,45,50] and the references therein). In particular, the "no read" and "no write" security notions for access control encryption are inspired by the "no read-up" and "no write-down" security policies first introduced in the seminal work of Bell and LaPadula [7]. In this work, we focus on designing cryptographic solutions for information flow control.

Numerous cryptographic primitives, starting with identity-based encryption [55,15,22], and progressing to attribute-based encryption [52,34,12], predicate encryption [17,36,40,43], and finally, culminating with functional encryption [51,16,44], have focused on ways of enabling fine-grained *access* to encrypted data (i.e., impose policies on the decryption capabilities of users in a system). Access control encryption seeks to simultaneously enforce policies on both the encryption capabilities of the sender as well as the decryption capabilities of the receiver.

A key challenge in access control encryption (and how it differs from traditional notions of functional encryption) is in preventing corrupt senders from communicating (covertly or otherwise) with unauthorized recipients. One way of viewing these goals is as a mechanism for protecting against steganography techniques [35]. Recent works on cryptographic reverse firewalls [42,26] have looked at preventing compromised or malicious software from leaking sensitive information. Raykova et al. [47] studied the problem of access control for outsourced data. Their goal was to hide access patterns from the cloud and preventing corrupt writers from updating files that they are not authorized to update. Their work considers a covert security model where malicious writers are caught; in contrast, with ACE, we require the stronger guarantee that communication between corrupt senders and unauthorized receivers are completely blocked.

Also related to access control encryption is the recent line of work on sanitizable signatures [3,20,27]. These works study the case where an intermediate party can sanitize messages and signatures that are sent over a channel while learning minimal information about the messages and signatures. The notion of

sanitizable signatures is conceptually different from that of ACE since sanitizable signatures are not designed to prevent corrupt senders from leaking information to corrupt receivers.

2 Preliminaries

For $n \geq 1$, we write $[n]$ to denote the set of integers $\{1, \ldots, n\}$. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is a sampled from $\mathcal{D}$. For a finite set S, we write $x \xleftarrow{\text{R}} S$ to denote that x is sampled uniformly at random from S. For a randomized function f, we write $f(x; r)$ to denote an evaluation of f using randomness r. Unless otherwise noted, we always write λ for the security parameter. We say a function $f(\lambda)$ is negligible in the security parameter λ (denoted $f(\lambda) = \mathsf{negl}(\lambda)$) if $f(\lambda) = o(1/\lambda^c)$ for all $c \in \mathbb{N}$. We write $f(\lambda) = \mathsf{poly}(\lambda)$ to denote that f is a (fixed) polynomial in λ. An algorithm is efficient if it runs in polynomial time in the length of its input. For two ensembles of distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, we write $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$ if the two distributions are computationally indistinguishable (that is, no efficient algorithm can distinguish $\mathcal{D}_1$ from $\mathcal{D}_2$ except with negligible probability).

We now formally define the tools we need to build our ACE scheme. Due to space limitations, we defer the standard definitions of a digital signature scheme and predicate encryption scheme to the full version of this paper [38]. In Section 2.1, we review the notion of functional encryption for randomized functionalities, and in Section 2.2, we introduce the notion of an access control encryption scheme.

2.1 Functional Encryption for Randomized Functionalities

Functional encryption (FE) [51,16,44] is a generalization of predicate encryption. In an FE scheme, secret keys are associated with functions and ciphertexts are associated with messages. Given a secret key sk_f for a (deterministic) function f and a ciphertext ct_x encrypting a value x, the decryption function in an FE scheme outputs $f(x)$. The security guarantee roughly states that sk_f and ct_x together reveal $f(x)$, and nothing more. Alwen et al. [2] and Goyal et al. [33] extended the notion of functional encryption to include support for *randomized* functionalities (i.e., secret keys are associated with randomized functions). Subsequently, Komargodski et al. [39], as well as Agrawal and Wu [1] showed how to generically transform FE schemes that support deterministic functions into schemes that support randomized functions; the former transformation [39] applies in the secret-key setting while the latter [1] applies in the public-key setting.

Syntax. We now give the formal definition of a functional encryption for randomized functionalities in the public-key setting. Our definitions are adapted from those in [33,1]. A functional encryption for randomized functionalities for a function family $\mathcal{F}$ over a domain $\mathcal{X}$, range $\mathcal{Y}$, and randomness space $\mathcal{R}$ is a tuple

of algorithms $\Pi_{\mathsf{rFE}} = (\mathsf{rFE.Setup}, \mathsf{rFE.KeyGen}, \mathsf{rFE.Encrypt}, \mathsf{rFE.Decrypt})$ with the following properties:

- $\mathsf{rFE.Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$: On input the security parameter λ, the setup algorithm outputs the public parameters pp and the master secret key msk.
- $\mathsf{rFE.KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$: On input the master secret key msk and the description of a (possibly randomized) function $f\colon \mathcal{X} \to \mathcal{Y}$, the key-generation algorithm outputs a secret key sk_f.
- $\mathsf{rFE.Encrypt}(\mathsf{pp}, x) \to \mathsf{ct}_x$: On input the public parameters pp and a message $x \in \mathcal{X}$, the encryption algorithm outputs a ciphertext ct_x.
- $\mathsf{rFE.Decrypt}(\mathsf{sk}, \mathsf{ct}) \to y$: On input a secret key sk, and a ciphertext ct, the decryption algorithm outputs a value $y \in \mathcal{Y} \cup \{\perp\}$.

Correctness. The correctness property for an FE scheme that supports randomized functionalities states that given a secret key sk_f for a randomized function f and a ciphertext ct_x encrypting a value x, the decryption function $\mathsf{rFE.Decrypt}(\mathsf{sk}_f, \mathsf{ct}_x)$ outputs a random draw from the output distribution of $f(x)$. Moreover, when multiple function keys are applied to multiple ciphertexts, decryption should output an *independent* draw from the output distribution for each ciphertext-key pair. This property should hold even given the public parameters as well as the function keys for the function encryption scheme. We give the formal definition below:

Definition 2.1 (Correctness [33,1, adapted]). *A functional encryption scheme for randomized functionalities* $\Pi_{\mathsf{rFE}} = (\mathsf{rFE.Setup}, \mathsf{rFE.KeyGen}, \mathsf{rFE.Encrypt}, \mathsf{rFE.Decrypt})$ *over a message space* $\mathcal{X}$ *for a (randomized) function family* $\mathcal{F}$ *(operating over a randomness space* $\mathcal{R}$*) is correct if for every polynomial* $n = n(\lambda)$*, every collection of functions* $(f_1, \ldots, f_n) \in \mathcal{F}^n$*, and every collection of messages* $(x_1, \ldots, x_n) \in \mathcal{X}^n$*, setting* $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{rFE.Setup}(1^\lambda)$*,* $\mathsf{sk}_i \leftarrow \mathsf{rFE.KeyGen}(\mathsf{msk}, f_i)$*,* $\mathsf{ct}_j \leftarrow \mathsf{rFE.Encrypt}(\mathsf{pp}, x_j)$*, and* $r_{i,j} \xleftarrow{\mathrm{R}} \mathcal{R}$ *for* $i, j \in [n]$*, the following two distributions are computationally indistinguishable:*

$$\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]}, \{\mathsf{rFE.Decrypt}(\mathsf{sk}_i, \mathsf{ct}_j)\}_{i,j\in[n]}\right) \quad \textit{and} \quad \left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]}, \{f_i(x_j; r_{i,j})\}_{i,j\in[n]}\right).$$

Remark 2.2 (Weaker Correctness Notions). Existing constructions of functional encryption for randomized functionalities [33,1] consider a weaker correctness requirement that the joint distribution $\{\mathsf{rFE.Decrypt}(\mathsf{sk}_i, \mathsf{ct}_j)\}_{i,j\in[n]}$ be computationally indistinguishable from $\{f_i(x_j; r_{i,j})\}_{i,j\in[n]}$. In this work, we require the stronger property that these two distributions remain computationally indistinguishable even given the public parameters as well as the (honestly-generated) decryption keys. It is not difficult to see that existing constructions such as the Agrawal-Wu generic construction [1] satisfy this stronger correctness requirement.[3]

[3] Specifically, the generic construction of functional encryption for randomized functionalities from standard functional encryption in [1] uses a PRF key for derandomization.

Security. In this work, we use a simulation-based definition of security. Our access control encryption construction relies critically on our FE scheme providing robustness against *malicious* encrypters. This can be viewed as the analog of CCA-security in the context of public-key encryption [46], and is captured formally in the security game by giving the adversary access to a decryption oracle (much like in the CCA-security game). We give a simplified variant of the definition from [33,1] where the adversary is only allowed to issue key-queries before making challenge queries (i.e., the adversary is restricted to making *non-adaptive* key queries). In this non-adaptive setting, Gorbunov et al. [31] showed that security against an adversary who makes a single challenge query implies security against an adversary that makes a polynomial number of of challenge queries. This is the definition we use in this work. Additionally, for the decryption queries, we also consider the simplified setting of [33] where the adversary can only submit a *single* ciphertext on each decryption query.[4] We now give the formal definition:

Definition 2.3 (q-NA-SIM Security [33,1, adapted]). *Let $\Pi_{\mathsf{rFE}} = (\mathsf{rFE.Setup}, \mathsf{rFE.KeyGen}, \mathsf{rFE.Encrypt}, \mathsf{rFE.Decrypt})$ be a functional encryption scheme for randomized functionalities over a message space $\mathcal{X}$ for a (randomized) function family $\mathcal{F}$ (with randomness space $\mathcal{R}$). We say that Π_{rFE} is q-NA-SIM-secure against malicious encrypters if there exists an efficient (stateful) simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4)$ such that for all efficient adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_1$ makes at most q key-generation queries, the outputs of the following two experiments are computationally indistinguishable:*

Experiment $\mathsf{Real}_{\Pi_{\mathsf{rFE}},\mathcal{A}}(1^\lambda)$:	**Experiment** $\mathsf{Ideal}_{\Pi_{\mathsf{rFE}},\mathcal{A},\mathcal{S}}(1^\lambda)$:
$(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{rFE.Setup}(1^\lambda)$	$(\mathsf{pp}, \mathsf{st}') \leftarrow \mathcal{S}_1(1^\lambda)$
$\mathsf{st} \leftarrow \mathcal{A}_1^{\mathcal{O}_1(\mathsf{msk},\cdot),\, \mathcal{O}_3(\mathsf{msk},\cdot,\cdot)}(1^\lambda, \mathsf{pp})$	$\mathsf{st} \leftarrow \mathcal{A}_1^{\mathcal{O}'_1(\mathsf{st}',\cdot),\, \mathcal{O}'_3(\mathsf{st}',\cdot,\cdot)}(1^\lambda, \mathsf{pp})$
$\alpha \leftarrow \mathcal{A}_2^{\mathcal{O}_2(\mathsf{pp},\cdot),\, \mathcal{O}_3(\mathsf{msk},\cdot,\cdot)}(\mathsf{st})$	$\alpha \leftarrow \mathcal{A}_2^{\mathcal{O}'_2(\mathsf{st}',\cdot),\, \mathcal{O}_3(\mathsf{st}',\cdot,\cdot)}(\mathsf{st})$
Output $(\{g\}, \{y\}, \alpha)$	**Output** $(\{g'\}, \{y'\}, \alpha)$

where the key-generation, encryption, and decryption oracles are defined as follows:

Real experiment $\mathsf{Real}_{\Pi_{\mathsf{rFE}},\mathcal{A}}(1^\lambda)$**:**

- ***Key-generation oracle:*** $\mathcal{O}_1(\mathsf{msk}, \cdot)$ *implements* $\mathsf{rFE.KeyGen}(\mathsf{msk}, \cdot)$.
- ***Encryption oracle:*** $\mathcal{O}_2(\mathsf{pp}, \cdot)$ *implements* $\mathsf{rFE.Encrypt}(\mathsf{pp}, \cdot)$.

In their construction, they secret share the PRF key across the ciphertext and the decryption key. By appealing to related-key security of the underlying PRF [13,11,8,9], the randomness used for function evaluation during decryption is computationally indistinguishable from a random string. Moreover, this holds even if one of the key-shares is known (in our setting, this is the key-share embedded within the decryption key).

[4] Subsequent work [1] showed how to extend the security definition to also capture adversaries that can induce correlations across multiple ciphertexts, but this strengthened definition is not necessary in our setting.

– ***Decryption oracle:*** *On input* (g, ct) *where* $g \in \mathcal{F}$ *and* $\mathsf{ct} \in \{0,1\}^*$*, the decryption oracle* $\mathcal{O}_3(\mathsf{msk}, \cdot, \cdot)$ *computes* $\mathsf{sk}_g \leftarrow \mathsf{rFE.KeyGen}(\mathsf{msk}, g)$ *and outputs* $y = \mathsf{rFE.Decrypt}(\mathsf{sk}_g, \mathsf{ct})$*. The (ordered) set* $\{g\}$ *consists of the set of functions that appear in the decryption queries of* $\mathcal{A}$ *and the (ordered) set* $\{y\}$ *consist of the responses of* $\mathcal{O}_3$*.*

Ideal experiment $\mathsf{Ideal}_{\Pi_{\mathsf{rFE}},\mathcal{A},\mathcal{S}}(1^\lambda)$**:**

– ***Key-generation oracle:*** *On input a function* $f \in \mathcal{F}$*, the ideal key-generation oracle* $\mathcal{O}'_1$ *computes* $(\mathsf{sk}'_f, \mathsf{st}') \leftarrow \mathcal{S}_2(\mathsf{st}', f)$*, and returns* sk'_f*. The updated simulator state* st' *is carried over to future invocations of the simulator.*
– ***Encryption oracle:*** *On input a message* $x \in \mathcal{X}$*, the ideal encryption oracle* $\mathcal{O}'_2$ *samples* $r_1, \ldots, r_q \xleftarrow{\text{R}} \mathcal{R}$*, and sets* $y_i = f_i(x; r_i)$ *for* $i \in [q]$*, where* f_i *is the* i^{th} *key-generation query* $\mathcal{A}_1$ *made to the key-generation oracle. The oracle computes* $(\mathsf{ct}', \mathsf{st}') \leftarrow \mathcal{S}_3(\mathsf{st}', \{y_i\}_{i \in [q]})$ *and returns* ct'*.*
– ***Decryption oracle:*** *On input* (g', ct') *where* $g' \in \mathcal{F}$ *and* $\mathsf{ct}' \in \{0,1\}^*$*, the ideal decryption oracle* $\mathcal{O}'_3$ *invokes the simulator algorithm* $(x, \mathsf{st}') \leftarrow \mathcal{S}_4(\mathsf{st}', \mathsf{ct}')$*, where* $x \in \mathcal{X} \cup \{\perp\}$*. If* $x \neq \perp$*, the oracle samples* $r \xleftarrow{\text{R}} \mathcal{R}$ *and replies with* $g'(x; r)$*. Otherwise, if* $x = \perp$*, the oracle replies with* $\perp$*. The (ordered) set* $\{g'\}$ *denotes the functions in the decryption queries of* $\mathcal{A}$ *and* $\{y'\}$ *denotes the outputs of* $\mathcal{O}'_3$*.*

2.2 Access Control Encryption (ACE)

In this section, we review the definition of *access control encryption* (ACE) [23,28,57]. An access control encryption scheme over an identity space $\mathcal{I}$, a message space $\mathcal{M}$, and a ciphertext space $\mathcal{C}$ is defined by a tuple of algorithms $\Pi_{\mathsf{ACE}} = (\mathsf{ACE.Setup}, \mathsf{ACE.EKGen}, \mathsf{ACE.DKGen}, \mathsf{ACE.Encrypt}, \mathsf{ACE.Sanitize}, \mathsf{ACE.Decrypt})$ with the following properties:

– $\mathsf{ACE.Setup}(1^\lambda, \pi) \to (\mathsf{sank}, \mathsf{msk})$: On input a security parameter λ and an access control policy $\pi \colon \mathcal{I} \times \mathcal{I} \to \{0,1\}$, the setup algorithm outputs the sanitizer key sank and the master secret key msk.
– $\mathsf{ACE.EKGen}(\mathsf{msk}, i) \to \mathsf{ek}_i$: On input the master secret key msk and a sender identity $i \in \mathcal{I}$, the encryption key-generation algorithm outputs an encryption key ek_i.
– $\mathsf{ACE.DKGen}(\mathsf{msk}, j) \to \mathsf{dk}_j$: On input the master secret key msk, and a receiver identity $j \in \mathcal{I}$, the decryption key-generation algorithm returns a decryption key dk_j.
– $\mathsf{ACE.Encrypt}(\mathsf{ek}, m) \to \mathsf{ct}$: On input an encryption key ek, and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.[5]

[5] Note that we do not require that $\mathsf{ct} \in \mathcal{C}$. In particular, the ciphertexts output by the encryption algorithm can be syntactically different from those output by the sanitize algorithm. To simplify the notation, we only explicitly model the ciphertexts space $\mathcal{C}$ corresponding to those produced by the $\mathsf{ACE.Sanitize}$ algorithm.

– $\mathsf{ACE.Sanitize}(\mathsf{sank}, \mathsf{ct}) \to \mathsf{ct}'$: On input the sanitizer key sank and a ciphertext ct, the sanitize algorithm outputs a ciphertext $\mathsf{ct}' \in \mathcal{C} \cup \{\perp\}$.
– $\mathsf{ACE.Decrypt}(\mathsf{dk}, \mathsf{ct}') \to m'$: On input a decryption key dk and a ciphertext $\mathsf{ct}' \in \mathcal{C}$, the decryption algorithm outputs a message $m' \in \mathcal{M} \cup \{\perp\}$.

Definition 2.4 (Correctness [23]). *An ACE scheme* $\Pi_{\mathsf{ACE}} = (\mathsf{ACE.Setup}, \mathsf{ACE.EKGen}, \mathsf{ACE.DKGen}, \mathsf{ACE.Encrypt}, \mathsf{ACE.Sanitize}, \mathsf{ACE.Decrypt})$ *over an identity space* $\mathcal{I}$ *and a message space* $\mathcal{M}$ *is correct if for all messages* $m \in \mathcal{M}$*, all policies* $\pi\colon \mathcal{I} \times \mathcal{I} \to \{0,1\}$*, and all identities* $i, j \in \mathcal{I}$ *where* $\pi(i,j) = 1$*, setting* $(\mathsf{sank}, \mathsf{msk}) \leftarrow \mathsf{ACE.Setup}(1^\lambda, \pi)$*,* $\mathsf{ek}_i \leftarrow \mathsf{ACE.EKGen}(\mathsf{msk}, i)$*,* $\mathsf{dk}_j \leftarrow \mathsf{ACE.DKGen}(\mathsf{msk}, j)$*, we have that*

$$\Pr[\mathsf{ACE.Decrypt}(\mathsf{dk}_j, \mathsf{ACE.Sanitize}(\mathsf{sank}, \mathsf{ACE.Encrypt}(\mathsf{ek}_i, m))) = m] = 1 - \mathsf{negl}(\lambda).$$

Security definitions. Damgård et al. [23] introduced two security notions for an ACE scheme: the *no-read rule* and the *no-write rule*. The no-read rule captures the property that only the intended recipients of a message (namely, those authorized to decrypt it) should be able to learn anything about the message. In particular, a subset of unauthorized receivers should be unable to combine their respective decryption keys to learn something about a ciphertext they are not authorized to decrypt. Moreover, this property should hold even if the recipients collude with the sanitizer. The no-write rule captures the property that a sender can only encrypt messages to receivers that it is authorized to do so. Specifically, no sender with identity i should be able to form a ciphertext that can be decrypted by a receiver with identity j where $\pi(i,j) = 0$. Furthermore, this property should hold even when multiple senders and receivers collude. We now review the formal definitions introduced in [23].

Definition 2.5 (No-Read Rule [23]). *Let* $\Pi_{\mathsf{ACE}} = (\mathsf{ACE.Setup}, \mathsf{ACE.EKGen}, \mathsf{ACE.DKGen}, \mathsf{ACE.Encrypt}, \mathsf{ACE.Sanitize}, \mathsf{ACE.Decrypt})$ *be an ACE scheme over an identity space* $\mathcal{I}$ *and a message space* $\mathcal{M}$*. Let* $\mathcal{A}$ *be an efficient adversary and* $\pi\colon \mathcal{I} \times \mathcal{I} \to \{0,1\}$ *be an access control policy. For a security parameter* λ *and a bit* $b \in \{0,1\}$*, we define the no-read rule experiment* $\mathsf{Expt}^{(\mathsf{Read})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda, b)$ *as follows. The challenger first samples* $(\mathsf{sank}, \mathsf{msk}) \leftarrow \mathsf{ACE.Setup}(1^\lambda, \pi)$*, and gives the sanitizer key* sank *to* $\mathcal{A}$*. Then,* $\mathcal{A}$ *is given access to the following oracles:*

– ***Encryption oracle.*** *On input a message* $m \in \mathcal{M}$ *and a sender identity* $i \in \mathcal{I}$*, the challenger responds with a ciphertext* $\mathsf{ct} \leftarrow \mathsf{ACE.Encrypt}(\mathsf{ACE.EKGen}(\mathsf{msk}, i), m)$*.*
– ***Encryption key-generation oracle.*** *On input a sender identity* $i \in \mathcal{I}$*, the challenger responds with an encryption key* $\mathsf{ek}_i \leftarrow \mathsf{ACE.EKGen}(\mathsf{msk}, i)$*.*
– ***Decryption key-generation oracle.*** *On input a receiver identity* $j \in \mathcal{I}$*, the challenger responds with a decryption key* $\mathsf{dk}_j \leftarrow \mathsf{ACE.DKGen}(\mathsf{msk}, j)$*.*
– ***Challenge oracle.*** *On input a pair of messages* $(m_0, m_1) \in \mathcal{M} \times \mathcal{M}$ *and a pair of sender indices* $(i_0, i_1) \in \mathcal{I} \times \mathcal{I}$*, the challenger responds with* $\mathsf{ACE.Encrypt}(\mathsf{ACE.EKGen}(\mathsf{msk}, i_b), m_b)$*.*

At the end of the experiment, adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment. An adversary $\mathcal{A}$ is admissible for the no-read rule security game if for all queries $j \in \mathcal{I}$ that $\mathcal{A}$ makes to the receiver key-generation oracle, $\pi(i_0, j) = 0 = \pi(i_1, j)$. We say that Π_{ACE} satisfies the no-read rule if for all policies $\pi \colon \mathcal{I} \times \mathcal{I} \to \{0,1\}$, and all efficient and admissible adversaries $\mathcal{A}$,

$$\left|\Pr\left[\mathsf{Expt}^{(\mathsf{Read})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda, 0) = 0\right] - \Pr\left[\mathsf{Expt}^{(\mathsf{Read})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}](\lambda, 1) = 1\right]\right| = \mathsf{negl}(\lambda).$$

Remark 2.6 (Sender Anonymity). The definition of the no-read rule given in [23] also imposes the stronger requirement of *sender anonymity*, which guarantees the anonymity of the sender even against adversaries that are able to decrypt the ciphertext. In contrast, our definition only ensures sender anonymity (in addition to message privacy) against a coalition of receivers that *cannot* decrypt the challenge ciphertext. This is akin to the notion of "weak attribute-hiding" in the context of predicate encryption [43,40], and was also the notion considered in [28] for building ACE for restricted classes of functionalities.

Definition 2.7 (No-Write Rule [23]). *Let $\Pi_{\mathsf{ACE}} = (\mathsf{ACE.Setup}, \mathsf{ACE.EKGen}, \mathsf{ACE.DKGen}, \mathsf{ACE.Encrypt}, \mathsf{ACE.Sanitize}, \mathsf{ACE.Decrypt})$ be an ACE scheme over an identity space $\mathcal{I}$ and a message space $\mathcal{M}$. Let $\mathcal{A}$ be an efficient adversary, and let $\pi \colon \mathcal{I} \times \mathcal{I} \to \{0,1\}$ be an access control policy. For a security parameter λ and a bit $b \in \{0,1\}$, we define the no-write rule experiment $\mathsf{Expt}^{(\mathsf{Write})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda, b)$ as follows. The challenger begins by sampling $(\mathsf{sank}, \mathsf{msk}) \leftarrow \mathsf{ACE.Setup}(1^\lambda, \pi)$. Then, $\mathcal{A}$ is given access to the following oracles:*

- ***Encryption oracle.*** *On input a message $m \in \mathcal{M}$ and a sender identity $i \in \mathcal{I}$, the challenger responds by first computing $\mathsf{ek}_i \leftarrow \mathsf{ACE.EKGen}(\mathsf{msk}, i)$ and returning $\mathsf{ACE.Sanitize}(\mathsf{sank}, \mathsf{ACE.Encrypt}(\mathsf{ek}_i, m))$.*
- ***Encryption key-generation oracle.*** *On input a sender index $i \in \mathcal{I}$, the challenger responds with an encryption key $\mathsf{ek}_i \leftarrow \mathsf{ACE.EKGen}(\mathsf{msk}, i)$.*
- ***Decryption key-generation oracle.*** *On input a receiver index $j \in \mathcal{I}$, the challenger responds with a decryption key $\mathsf{dk}_j \leftarrow \mathsf{ACE.DKGen}(\mathsf{msk}, j)$.*
- ***Challenge oracle.*** *On input a ciphertext $\mathsf{ct}^* \in \{0,1\}^*$ and a sender identity $\mathsf{id}^* \in \mathcal{I}$, the challenger sets $\mathsf{ct}_0 = \mathsf{ct}^*$. Then, the challenger samples $m' \xleftarrow{\mathrm{R}} \mathcal{M}$, computes $\mathsf{ct}_1 \leftarrow \mathsf{ACE.Encrypt}(\mathsf{ACE.EKGen}(\mathsf{msk}, \mathsf{id}^*), m')$, and responds with $\mathsf{ACE.Sanitize}(\mathsf{sank}, \mathsf{ct}_b)$.*

At the end of the experiment, adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment. An adversary $\mathcal{A}$ is admissible for the no-write rule security game if the following conditions hold:

- *The adversary $\mathcal{A}$ makes at most one query to the challenge oracle.*[6]

[6] We impose this restriction to simplify the security definition. A standard hybrid argument shows that security against an adversary that makes a single challenge query implies security against one that makes multiple challenge queries.

– *For all identities $i \in \mathcal{I}$ that $\mathcal{A}$ submits to the encryption key-generation oracle prior to its challenge and all identities $j \in \mathcal{I}$ that $\mathcal{A}$ submits to the decryption key-generation oracle, $\pi(i, j) = 0$.*
– *The adversary $\mathcal{A}$ makes an encryption key-generation query on the challenge identity $\mathsf{id}^* \in \mathcal{I}$ prior to making its challenge query.*

We say that Π_{ACE} satisfies the no-write rule if for all policies $\pi : \mathcal{I} \times \mathcal{I} \to \{0, 1\}$, and all efficient and admissible adversaries $\mathcal{A}$,

$$\left|\Pr\left[\mathsf{Expt}^{(\mathsf{Write})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda, 0) = 0\right] - \Pr\left[\mathsf{Expt}^{(\mathsf{Write})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda, 1) = 1\right]\right| = \mathsf{negl}(\lambda).$$

3 Generic Construction of Access Control Encryption

In this section, we show how to generically construct access control encryption for general policies from a digital signature scheme, a predicate encryption scheme, and a general-purpose functional encryption scheme for randomized functionalities. Then, in Section 3.1, we describe our concrete instantiation of an ACE scheme that supports arbitrary policies from standard assumptions.

Construction 3.1 *Let $\mathcal{I}$ be the identity space and $\mathcal{M}$ be the message space. Our access control encryption for general access policies relies on the following primitives:*

– *Let $\Pi_{\mathsf{Sig}} = (\mathsf{Sig.Setup}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ be a signature scheme with message space $\mathcal{I}$. Let $\mathcal{T}$ denote the space of signatures output by the* $\mathsf{Sig.Sign}$ *algorithm.*
– *Let $\Pi_{\mathsf{PE}} = (\mathsf{PE.Setup}, \mathsf{PE.KeyGen}, \mathsf{PE.Encrypt}, \mathsf{PE.Decrypt})$ be a (public-key) predicate encryption scheme with attribute space $\mathcal{I}$ and message space $\mathcal{M}$. Let $\mathcal{C}$ denote the ciphertext space for Π_{PE}, and let $\mathcal{R}$ denote the space for the encryption randomness for* $\mathsf{PE.Encrypt}$ *(namely, the space of values from which the randomness used in* $\mathsf{PE.Encrypt}$ *is sampled).*
– *Let $\Pi_{\mathsf{rFE}} = (\mathsf{rFE.Setup}, \mathsf{rFE.KeyGen}, \mathsf{rFE.Encrypt}, \mathsf{rFE.Decrypt})$ be a general-purpose public-key functional encryption scheme for randomized functionalities (with security against malicious encrypters) with domain $\mathcal{I} \times \mathcal{T} \times \mathcal{M}$, range $\mathcal{C}$, and randomness space $\mathcal{R}$.*

We construct the ACE scheme $\Pi_{\mathsf{ACE}} = (\mathsf{ACE.Setup}, \mathsf{ACE.EKGen}, \mathsf{ACE.DKGen}, \mathsf{ACE.Encrypt}, \mathsf{ACE.Sanitize}, \mathsf{ACE.Decrypt})$ as follows:

– $\mathsf{ACE.Setup}(1^\lambda, \pi)$*: On input the security parameter λ and a policy $\pi : \mathcal{I} \times \mathcal{I} \to \{0, 1\}$, the setup algorithm samples* $(\mathsf{Sig.vk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$, $(\mathsf{PE.pp}, \mathsf{PE.msk}) \leftarrow \mathsf{PE.Setup}(1^\lambda)$, *and* $(\mathsf{rFE.pp}, \mathsf{rFE.msk}) \leftarrow \mathsf{rFE.Setup}(1^\lambda)$. *Next, it defines the function $F_{\mathsf{Sig.vk},\mathsf{PE.pp}} : \mathcal{I} \times \mathcal{T} \times \mathcal{M} \to \mathcal{C}$ as follows:*

$$F_{\mathsf{Sig.vk},\mathsf{PE.pp}}(i, \sigma, m; r) = \begin{cases} \mathsf{PE.Encrypt}(\mathsf{PE.pp}, i, m; r) & \text{if } \mathsf{Sig.Verify}(\mathsf{Sig.vk}, i, \sigma) = 1 \\ \perp & \text{otherwise.} \end{cases}$$

Then, it generates a decryption key $\mathsf{rFE.sk}_F \leftarrow \mathsf{rFE.KeyGen}(\mathsf{rFE.msk}, F_{\mathsf{Sig.vk},\mathsf{PE.pp}})$. *Finally, it outputs the sanitizer key* $\mathsf{sank} = \mathsf{rFE.sk}_F$ *and the master secret key*

$$\mathsf{msk} = (\pi, \mathsf{Sig.sk}, \mathsf{PE.msk}, \mathsf{rFE.pp}).$$

- $\mathsf{ACE.EKGen}(\mathsf{msk}, i)$*: On input the master secret key* $\mathsf{msk} = (\pi, \mathsf{Sig.sk}, \mathsf{PE.msk}, \mathsf{rFE.pp})$*, and an identity* $i \in \mathcal{I}$*, the encryption key-generation algorithm constructs a signature* $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, i)$ *and outputs* $\mathsf{ek}_i = (\mathsf{rFE.pp}, i, \sigma)$.
- $\mathsf{ACE.DKGen}(\mathsf{msk}, j)$*: On input the master secret key* $\mathsf{msk} = (\pi, \mathsf{Sig.sk}, \mathsf{PE.msk}, \mathsf{rFE.pp})$*, and an identity* $j \in \mathcal{I}$*, the decryption key-generation algorithm generates a key* $\mathsf{PE.sk} \leftarrow \mathsf{PE.KeyGen}(\mathsf{PE.msk}, f_{\pi,j})$ *where* $f_{\pi,j}(i) \colon \mathcal{I} \to \{0,1\}$ *is defined as* $f_{\pi,j}(i) = \pi(i,j)$*, and outputs* $\mathsf{dk}_j = \mathsf{PE.sk}$.
- $\mathsf{ACE.Encrypt}(\mathsf{ek}_i, m)$*: On input the encryption key* $\mathsf{ek}_i = (\mathsf{rFE.pp}, i, \sigma)$ *and a message* $m \in \mathcal{M}$*, the encryption algorithm outputs* $\mathsf{rFE.Encrypt}(\mathsf{rFE.pp}, (i, \sigma, m))$.
- $\mathsf{ACE.Sanitize}(\mathsf{sank}, \mathsf{ct})$*: On input the sanitizer key* $\mathsf{sank} = \mathsf{rFE.sk}_F$ *and a ciphertext* ct*, the sanitize algorithm outputs* $\mathsf{rFE.Decrypt}(\mathsf{rFE.sk}_F, \mathsf{ct})$.
- $\mathsf{ACE.Decrypt}(\mathsf{dk}_j, \mathsf{ct}')$*: On input a decryption key* $\mathsf{dk}_j = \mathsf{PE.sk}$ *and a ciphertext* ct'*, the decryption algorithm outputs* $\mathsf{PE.Decrypt}(\mathsf{PE.sk}, \mathsf{ct}')$.

We now state that our main correctness and security theorems. Specifically, we show that assuming correctness and security of the underlying primitive Π_{Sig}, Π_{PE}, and Π_{FE}, our access control encryption scheme satisfies correctness (Definition 2.4), no-read security (Definition 2.5), and no-write security (Definition 2.7). We give the proof of Theorem 3.2 in the full version [38] and the proofs of Theorems 3.3 and 3.4 in Sections 3.2 and 3.3, respectively. We conclude this subsection with a remark comparing our construction to the Damgård et al. [23] construction of ACE from sanitizable FE.

Theorem 3.2 (Correctness). *Suppose* Π_{Sig} *is a correct signature scheme,* Π_{PE} *is a correct predicate encryption scheme, and* Π_{FE} *is a correct functional encryption scheme for randomized functionalities (Definition 2.1). Then, the access control encryption scheme from Construction 3.1 is correct (Definition 2.4).*

Theorem 3.3 (No-Read Rule). *Suppose* Π_{Sig} *is perfectly correct,* Π_{PE} *is a secure predicate encryption scheme and* Π_{rFE} *is an* 1-*NA-SIM-secure functional encryption scheme for randomized functionalities (Definition 2.3). Then, the access control encryption scheme from Construction 3.1 satisfies the no-read rule (Definition 2.5).*

Theorem 3.4 (No-Write Rule). *If* Π_{Sig} *is existentially unforgeable,* Π_{PE} *is a secure predicate encryption scheme, and* Π_{rFE} *is a* 1-*NA-SIM-secure functional encryption for randomized functionalities (Definition 2.3). Then, the access control encryption scheme from Construction 3.1 satisfies the no-write rule (Definition 2.7).*

Remark 3.5 (Comparison with Sanitizable FE). The high-level schema of our access control encryption scheme bears some similarities to the ACE construction

from sanitizable functional encryption in [23]. Here, we highlight some of the key differences between our construction and that of [23]. In [23], the sanitizer key is used only to test whether a particular ciphertext is valid or not. After validating the certificate, the sanitizer relies on the *algebraic* structure of the sanitizable FE scheme to *re-randomize* the ciphertext. In contrast, in our construction, the sanitizer actually performs a *re-encryption* of the incoming ciphertext under a *different* (predicate) encryption scheme, and moreover, the validation procedure (that the ciphertext originated from a valid sender) is embedded within the re-encryption key possessed by the sanitizer. As such, our construction only requires us to issue a single functional encryption key to the sanitizer. This means that we can base our construction on standard cryptographic assumptions. While it may be possible to build sanitizable FE from an FE scheme that supports randomized functionalities, it seems difficult to reduce security to standard assumptions (because the existing general-purpose FE schemes from standard assumptions [51,31,30] remain secure only if we give out an *a priori* bounded number of decryption keys). Thus, using re-encryption rather than re-randomization offers qualitatively better properties that enables a construction that does not rely on strong assumptions like indistinguishability obfuscation.

3.1 Concrete Instantiations

In this section, we describe one candidate instantiation of Construction 3.1 that yields an access control encryption scheme for arbitrary policies from standard assumptions. All of our primitives can be built from standard assumptions, namely the decisional Diffie-Hellman assumption (DDH) [14], the RSA assumption (RSA) [49], and the learning with errors assumption (LWE) [48]. The DDH and RSA assumptions are needed to leverage the generic construction of functional encryption for randomized functionalities from standard functional encryption (for deterministic functionalities) in [1]. The remaining primitives can be built from LWE. We now describe one possible instantiation of the primitives in Construction 3.1:

- The signature scheme Π_{Sig} can be instantiated using the standard-model construction of Cash et al. [21] based on LWE. Note that because our construction makes non-black-box use of the underlying signature scheme (in particular, we need to issue an FE key that performs signature verification), we are unable to instantiate our construction with a signature scheme that relies on a random oracle.
- The (general-purpose) predicate encryption scheme Π_{PE} can be instantiated using the construction of Gorbunov et al. [32] based on the LWE assumption.
- The (general-purpose) 1-NA-SIM-secure FE scheme Π_{rFE} for randomized functionalities that provides security against malicious encrypters can be instantiated by applying the Agrawal-Wu deterministic-to-randomized transformation [1] to a 1-NA-SIM-secure FE scheme for deterministic functionalities. The underlying 1-NA-SIM-secure FE scheme can in turn be based on any public-key encryption [31] or on the LWE assumption [30]. Applying

the deterministic-to-randomized transformation to the former yields an FE scheme for randomized functionalities from the DDH and RSA assumptions (cf. [1, Corollary 5.5]), while applying the transformation to the latter yields an FE scheme based on the DDH, RSA, and LWE assumptions.

Putting the pieces together, we obtain the following corollary to Theorems 3.3 and 3.4:

Corollary 3.6. *Under standard assumptions (namely, the DDH, RSA, and LWE assumptions), there exists an access control scheme for general policies over arbitrary identity spaces $\mathcal{I} = \{0,1\}^n$ where $n = \mathsf{poly}(\lambda)$ that satisfies the no-read and no-write security properties.*

3.2 Proof of Theorem 3.3

Our proof proceeds via a sequence of hybrid experiments between an adversary $\mathcal{A}$ and a challenger. First, fix an access control policy $\pi\colon \mathcal{I} \times \mathcal{I} \to \{0,1\}$. We now define our sequence of hybrid experiments:

- Hyb_0: This is the ACE security experiment $\mathsf{Expt}^{(\mathsf{Read})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda, 0)$ from Definition 2.5. Specifically, at the beginning of the game, the challenger samples keys $(\mathsf{Sig.vk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$, $(\mathsf{PE.pp}, \mathsf{PE.msk}) \leftarrow \mathsf{PE.Setup}(1^\lambda)$, and $(\mathsf{rFE.pp}, \mathsf{rFE.msk}) \leftarrow \mathsf{rFE.Setup}(1^\lambda)$. It then generates the sanitizer key $\mathsf{rFE.sk}_F \leftarrow \mathsf{rFE.KeyGen}(\mathsf{rFE.msk}, F_{\mathsf{Sig.vk},\mathsf{PE.pp}})$ and gives $\mathsf{sank} = \mathsf{rFE.sk}_F$ to the adversary. It sets $\mathsf{msk} = (\pi, \mathsf{Sig.sk}, \mathsf{PE.msk}, \mathsf{rFE.pp})$. During the query phase, the challenger answers the adversary's queries to the encryption and key-generation oracles by computing the encryption and key-generation algorithms exactly as in the real scheme. When the adversary makes a challenge oracle query with messages $(m_0, m_1) \in \mathcal{M} \times \mathcal{M}$ and identities $(i_0, i_1) \in \mathcal{I} \times \mathcal{I}$, the challenger responds with $\mathsf{ACE.Encrypt}(\mathsf{ACE.EKGen}(\mathsf{msk}, i_0), m_0)$.
- Hyb_1: Same as Hyb_0, except that the challenger uses the simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4,)$ for Π_{rFE} to construct the public parameters, the sanitizer key sank, and in replying to the adversary's challenge queries. Specifically, we make the following changes to the challenger:
  - **Setup:** At the beginning of the game, instead of sampling $\mathsf{rFE.pp}$ using $\mathsf{rFE.Setup}$, the challenger instead runs the simulation algorithm $(\mathsf{rFE.pp}, \mathsf{st}') \leftarrow \mathcal{S}_1(1^\lambda)$. For the sanitizer key, the challenger computes $\mathsf{rFE.sk}_F \leftarrow \mathcal{S}_2(\mathsf{st}', F_{\mathsf{Sig.vk},\mathsf{PE.pp}})$. It saves $\mathsf{rFE.pp}$ as part of the master secret key and gives $\mathsf{sank} = (\mathsf{rFE.sk}_F)$ to the adversary.
  - **Challenge queries:** When the adversary submits a challenge (m_0, m_1, i_0, i_1), the challenger first computes $\mathsf{ct}' \leftarrow \mathsf{PE.Encrypt}(\mathsf{PE.pp}, i_0, m_0)$. Then it replies to the adversary with the simulated ciphertext $\mathsf{ct} \leftarrow \mathcal{S}_3(\mathsf{st}', \mathsf{ct}')$.

  The encryption and key-generation queries are handled exactly as in Hyb_0.
- Hyb_2: Same as Hyb_1, except when answering challenge queries (m_0, m_1, i_0, i_1), the challenger instead computes $\mathsf{ct}' \leftarrow \mathsf{PE.Encrypt}(\mathsf{PE.pp}, i_1, m_1)$ and replies with the simulated ciphertext $\mathsf{ct} \leftarrow \mathcal{S}_3(\mathsf{st}', \mathsf{ct}')$.

– Hyb_3: Same as Hyb_2, except that the challenger constructs the public parameters $\mathsf{rFE.pp}$ and the sanitizer key sank as described in the real scheme. For challenge queries (m_0, m_1, i_0, i_1), the challenger replies with the ciphertext $\mathsf{ACE.Encrypt}(\mathsf{ACE.EKGen}(\mathsf{msk}, i_1), m_1)$. This corresponds to the ACE security experiment $\mathsf{Expt}^{(\mathsf{Read})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda, 1)$ from Definition 2.5.

We now argue that each pair of hybrid experiments are computationally indistinguishable. For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of Hyb_i. In the following, we implicitly assume that the adversary in each pair of hybrid arguments is admissible.

Lemma 3.7. *If Π_{Sig} is perfectly correct and Π_{rFE} is 1-NA-SIM-secure, then for all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between Hyb_0 and Hyb_1. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that can distinguish between $\mathsf{Real}_{\Pi_{\mathsf{rFE}},\mathcal{A}}(1^\lambda)$ and $\mathsf{Ideal}_{\Pi_{\mathsf{rFE}},\mathcal{A},\mathcal{S}}(1^\lambda)$. Algorithm $\mathcal{B}$ works as follows:

1. At the beginning of the game, algorithm $\mathcal{B}$ is given the public parameters $\mathsf{rFE.pp}$. It constructs the other components of the master secret key msk for the ACE scheme exactly as in Hyb_0 and Hyb_1.
2. Algorithm $\mathcal{B}$ makes a key-generation query for the function $F_{\mathsf{Sig.vk},\mathsf{PE.pp}}$ and receives a key $\mathsf{rFE.sk}_F$. It sets $\mathsf{sank} = \mathsf{rFE.sk}_F$ and gives $\mathsf{rFE.sk}_F$ to $\mathcal{A}$.
3. Algorithm $\mathcal{B}$ answers the encryption and key-generation queries exactly as in Hyb_0 and Hyb_1 (this is possible because these queries only rely on $\mathsf{rFE.pp}$).
4. Whenever $\mathcal{A}$ makes a challenge query (m_0, m_1, i_0, i_1), algorithm $\mathcal{B}$ computes a signature $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, i_0)$ and queries its encryption oracle on the value (i_0, σ, m_0) to obtain a challenge ciphertext ct. It gives ct to the adversary.
5. At the end of the game, algorithm $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

First, we note that $\mathcal{B}$ makes a single non-adaptive key query, so it is a valid adversary for the 1-NA-SIM security game. By construction, if the public parameters, the key-generation oracle and the encryption oracle are implemented according to $\mathsf{Real}_{\Pi_{\mathsf{rFE}},\mathcal{A}}(1^\lambda)$, then $\mathcal{B}$ perfectly simulates Hyb_0 for $\mathcal{A}$. We claim that if the public parameters, the key-generation oracle, and the encryption oracle are implemented according to $\mathsf{Ideal}_{\Pi_{\mathsf{rFE}},\mathcal{A},\mathcal{S}}(1^\lambda)$, then $\mathcal{B}$ perfectly simulates Hyb_1. It suffices to check that the challenge queries are correctly simulated.

– In Hyb_1, on a challenge query (m_0, m_1, i_0, i_1), the challenger responds by computing $\mathcal{S}_3(\mathsf{st}', \mathsf{ct}')$ where $\mathsf{ct}' \leftarrow \mathsf{PE.Encrypt}(\mathsf{PE.pp}, i_0, m_0)$.
– In the reduction, if the encryption oracle is implemented according to $\mathsf{Ideal}_{\Pi_{\mathsf{rFE}},\mathcal{A},\mathcal{S}}(1^\lambda)$, then $\mathcal{B}$'s response ct to a challenge query (m_0, m_1, i_0, i_1) is the output of $\mathcal{S}_3(\mathsf{st}', \mathsf{ct}')$, where $\mathsf{ct}' \leftarrow F_{\mathsf{Sig.vk},\mathsf{PE.pp}}(i_0, \sigma, m_0)$ and $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, i_0)$. By perfect correctness of Π_{Sig} and definition of $F_{\mathsf{Sig.vk},\mathsf{PE.pp}}$, the output distribution of $F_{\mathsf{Sig.vk},\mathsf{PE.pp}}(i_0, \sigma, m_0)$ is exactly a fresh encryption $\mathsf{PE.Encrypt}(\mathsf{PE.pp}, i_0, m_0)$.

We conclude that if the oracles are implemented according to $\mathsf{Ideal}_{\Pi_{\mathsf{rFE}},\mathcal{A},\mathcal{S}}(1^\lambda)$, then $\mathcal{B}$ perfectly simulates Hyb_1 for $\mathcal{A}$. The claim then follows by 1-NA-SIM security of Π_{rFE}. □

Lemma 3.8. *If Π_{PE} is secure, then for all efficient adversaries $\mathcal{A}$,*

$$|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between Hyb_1 and Hyb_2. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that can break the security of the predicate encryption scheme Π_{PE}. Algorithm $\mathcal{B}$ works as follows:

1. At the beginning of the game, $\mathcal{B}$ receives $\mathsf{PE.pp}$ from the predicate encryption challenger. It samples the parameters for the signature scheme as well as the parameters for the functional encryption scheme as described in Hyb_1 and Hyb_2 (in particular, the simulator uses the honest key-generation algorithm to sample the parameters for Π_{Sig} and uses the simulator $\mathcal{S}$ for Π_{rFE} to construct the parameters $\mathsf{rFE.pp}$). Algorithm $\mathcal{B}$ constructs the sanitizer key sank as in Hyb_1 and Hyb_2 (using $\mathsf{PE.pp}$), and gives sank to the adversary. It also defines msk as in the real scheme, with the exception that it leaves $\mathsf{PE.msk}$ unspecified.
2. During the query phase, $\mathcal{B}$ answers the encryption and encryption key-generation queries exactly as in Hyb_1 and Hyb_2 (these queries only depend on quantities known to $\mathcal{B}$). The decryption key-generation and challenge queries are handled as follows:
   - **Decryption key-generation oracle**: When $\mathcal{A}$ queries for a decryption key for an identity $j \in \mathcal{I}$, algorithm $\mathcal{B}$ submits the function $f_{\pi,j} : \mathcal{I} \to \{0,1\}$ (where $f_{\pi,j}(i) = \pi(i,j)$) to the key-generation oracle for the predicate encryption game, and receives the key $\mathsf{PE.sk}_{f_{\pi,j}}$. It gives $\mathsf{PE.sk}_{f_{\pi,j}}$ to $\mathcal{A}$.
   - **Challenge oracle**: When $\mathcal{A}$ makes its challenge query (m_0, m_1, i_0, i_1), algorithm $\mathcal{B}$ submits the pairs (i_0, m_0), (i_1, m_1) as its challenge query to the predicate encryption challenger and receives a ciphertext ct'. It runs the simulator $\mathsf{ct} \leftarrow \mathcal{S}_3(\mathsf{st}', \mathsf{ct}')$ and returns ct to $\mathcal{A}$.

Since $\mathcal{A}$ is admissible for the no-read rule security game, $\pi(i_0, j) = 0 = \pi(i_1, j)$ for all identities j that the adversary submits to the decryption key-generation oracle. This means that each function $f_{\pi,j}$ that $\mathcal{B}$ submits to the predicate encryption challenger satisfies $f_{\pi,j}(i_0) = 0 = f_{\pi,j}(i_1)$. Thus, $\mathcal{B}$ is admissible for the predicate encryption security game. By construction, if $\mathcal{B}$ is interacting according to $\mathsf{Expt}^{\mathsf{PE}}_{\Pi_{\mathsf{PE}},\mathcal{B}}(\lambda, 0)$, then $\mathcal{B}$ perfectly simulates Hyb_1 for $\mathcal{A}$, and if $\mathcal{B}$ is interacting according to $\mathsf{Expt}^{\mathsf{PE}}_{\Pi_{\mathsf{PE}},\mathcal{B}}(\lambda, 1)$, then $\mathcal{B}$ perfectly simulates Hyb_2 for $\mathcal{A}$. Thus, if $\mathcal{A}$ is able to distinguish between Hyb_1 and Hyb_2 with non-negligible advantage, then $\mathcal{B}$ is able to break the security of Π_{PE} with the same advantage. □

Lemma 3.9. *If Π_{Sig} is perfectly correct, and Π_{rFE} is 1-NA-SIM-secure, then for all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Follows by a similar argument as that used in the proof of Lemma 3.7. □

Combining Lemmas 3.7 through 3.9, we conclude that the ACE scheme in Construction 3.1 satisfies the no-read rule. □

3.3 Proof of Theorem 3.4

Our proof proceeds via a sequence of hybrid experiments between an adversary $\mathcal{A}$ and a challenger.

- Hyb_0: This is the ACE security experiment $\mathsf{Expt}^{(\mathsf{Write})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda, 0)$ from Definition 2.7. The challenger begins by sampling $(\mathsf{Sig.vk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.Setup}(1^\lambda)$, $(\mathsf{PE.pp}, \mathsf{PE.msk}) \leftarrow \mathsf{PE.Setup}(1^\lambda)$, and $(\mathsf{rFE.pp}, \mathsf{rFE.msk}) \leftarrow \mathsf{rFE.Setup}(1^\lambda)$. Then, it generates the decryption key $\mathsf{rFE.sk}_F \leftarrow \mathsf{rFE.KeyGen}(\mathsf{rFE.msk}, F_{\mathsf{Sig.vk},\mathsf{PE.pp}})$, and sets $\mathsf{sank} = \mathsf{rFE.sk}_F$ and $\mathsf{msk} = (\pi, \mathsf{Sig.sk}, \mathsf{PE.msk}, \mathsf{rFE.pp})$. During the query phase, the challenger answers the adversary's key-generation and encryption queries exactly as in the real scheme. When the adversary makes a challenge query on a ciphertext ct^* and an identity $\mathsf{id}^* \in \mathcal{I}$, the challenger responds with $\mathsf{ACE.Sanitize}(\mathsf{sank}, \mathsf{ct}^*)$.
- Hyb_1: Same as Hyb_0, except the challenger responds to the adversary's encryption queries with independently-generated predicate encryption ciphertexts. Specifically, for each encryption query on a message $m \in \mathcal{M}$ and identity $i \in \mathcal{I}$, the challenger responds with a fresh encryption $\mathsf{PE.Encrypt}(\mathsf{PE.pp}, i, m)$. The rest of the experiment remains unchanged.
- Hyb_2: Same as Hyb_1, except the challenger constructs the public parameters for the FE scheme, the sanitizer key, and its response to the challenge query using the simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4)$ for Π_{rFE} from Definition 2.3. Specifically, we make the following changes to the challenger:
  - **Setup:** At the beginning of the game, instead of sampling $\mathsf{rFE.pp}$ using $\mathsf{rFE.Setup}$, the challenger instead runs the simulation algorithm $(\mathsf{rFE.pp}, \mathsf{st}') \leftarrow \mathcal{S}_1(1^\lambda)$. For the sanitizer key, the challenger computes $\mathsf{rFE.sk}_F \leftarrow \mathcal{S}_2(\mathsf{st}', F_{\mathsf{Sig.vk},\mathsf{PE.pp}})$. The challenger samples $(\mathsf{Sig.vk}, \mathsf{Sig.sk})$ and $(\mathsf{PE.pp}, \mathsf{PE.msk})$ as in the real scheme.
  - **Challenge query:** For the challenge query $(\mathsf{ct}^*, \mathsf{id}^*)$, the challenger first invokes the simulator to obtain $y^* \leftarrow \mathcal{S}_4(\mathsf{st}', \mathsf{ct}^*)$. If $y^* \neq \perp$, it parses $y^* = (i^*, \sigma^*, m^*)$, and checks if $\mathsf{Sig.Verify}(\mathsf{Sig.vk}, i^*, \sigma^*) \stackrel{?}{=} 1$. If so, then the challenger returns $\mathsf{PE.Encrypt}(\mathsf{PE.pp}, i^*, m^*)$. In all other cases, the challenger outputs $\perp$.

  The rest of the experiment is identical to Hyb_1.
- Hyb_3: Same as Hyb_2, except the challenger aborts during the challenge phase if after computing $y^* \leftarrow \mathcal{S}_4(\mathsf{st}', \mathsf{ct}^*)$ and parsing $y^* = (i^*, \sigma^*, m^*)$, the following two conditions hold:
  - Adversary $\mathcal{A}$ did not previously make an encryption key-generation query for identity i^*.
  - $\mathsf{Sig.Verify}(\mathsf{Sig.vk}, i^*, \sigma^*) = 1$.

  Otherwise, the challenger proceeds as in Hyb_2.

- Hyb_4: Same as Hyb_3, except the challenger answers the challenge query with a sanitized encryption of a random message. Specifically, when the challenger receives a challenge query $(\mathsf{ct}^*, \mathsf{id}^*)$, it computes $y^* \leftarrow \mathcal{S}_4(\mathsf{st}', \mathsf{ct}^*)$ as usual and returns $\perp$ if $y^* = \perp$. Otherwise, it parses $y^* = (i^*, \sigma^*, m^*)$ and checks that $\mathsf{Sig.Verify}(\mathsf{Sig.vk}, i^*, \sigma^*) = 1$ (outputting $\perp$ if not). The challenger also checks the abort condition in Hyb_3. If all the checks pass, the challenger samples a message $m' \xleftarrow{\mathrm{R}} \mathcal{M}$ and returns $\mathsf{PE.Encrypt}(\mathsf{PE.pp}, \mathsf{id}^*, m')$ to the adversary. The rest of the experiment is unchanged.
- Hyb_5: Same as Hyb_4, except we remove the abort condition from the challenger.
- Hyb_6: Same as Hyb_5, except the challenger samples the public parameters for the FE scheme, the sanitizer key, and its response to the challenge query using the real algorithms Π_{rFE} rather than the simulator. In particular, when responding to the challenge query $(\mathsf{ct}^*, \mathsf{id}^*)$, the challenger responds with $\mathsf{rFE.Decrypt}(\mathsf{sank}, \mathsf{rFE.Encrypt}(\mathsf{rFE.pp}, (\mathsf{id}^*, \sigma, m')))$ where $m' \xleftarrow{\mathrm{R}} \mathcal{M}$ and σ is a signature on id^* under $\mathsf{Sig.vk}$.
- Hyb_7: Same as Hyb_6, except the challenger responds to the adversary's encryption queries honestly as in the real scheme instead of responding with independently generated predicate encryption ciphertexts. This corresponds to the ACE security experiment $\mathsf{Expt}^{(\mathsf{Write})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda, 1)$ from Definition 2.7.

Lemma 3.10. *If Π_{Sig} is perfectly correct and Π_{rFE} is correct, then for all efficient adversaries $\mathcal{A}$, we have that* $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. The only difference between Hyb_0 and Hyb_1 is the way the challenger responds to the adversary's encryption queries. First, let $\mathsf{sank} = \mathsf{rFE.sk}_F \leftarrow \mathsf{rFE.KeyGen}(\mathsf{rFE.pp}, F_{\mathsf{Sig.vk},\mathsf{PE.pp}})$ be the sanitizer key generated by the challenger at setup. Suppose the adversary makes Q encryption queries on message-identity pairs $(m_1, i_1), \ldots, (m_Q, i_Q)$. In Hyb_0, the challenger responds to each query (m_k, i_k) by first computing the signature $\sigma_k \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, i_k)$ and the ciphertext $\mathsf{ct}_k \leftarrow \mathsf{rFE.Decrypt}(\mathsf{rFE.sk}_F, \mathsf{rFE.Encrypt}(\mathsf{rFE.pp}, (i_k, \sigma_k, m_k)))$. By correctness of Π_{rFE}, we have that

$$\left(\mathsf{rFE.pp}, \mathsf{rFE.sk}_F, \{\mathsf{ct}_k\}_{k\in[Q]}\right) \overset{c}{\approx} \left(\mathsf{rFE.pp}, \mathsf{rFE.sk}_F, \{F_{\mathsf{Sig.vk},\mathsf{PE.pp}}(i_k, \sigma_k, m_k; r_k)\}_{k\in[Q]}\right),$$

where $r_k \xleftarrow{\mathrm{R}} \mathcal{R}$. Since σ_k is a signature on i_k, by perfect correctness of Π_{Sig} and definition of $F_{\mathsf{Sig.vk},\mathsf{PE.pp}}$, the output distribution of $F_{\mathsf{Sig.vk},\mathsf{PE.pp}}(i_k, \sigma_k, m_k; r_k)$ is precisely a fresh encryption $\mathsf{PE.Encrypt}(\mathsf{PE.pp}, i_k, m_k)$. This is the distribution in Hyb_1. Note that we include the sanitizer key $\mathsf{rFE.sk}_F$ in the joint distributions above because it is needed to simulate the response to the adversary's challenge query in Hyb_0 and Hyb_1. □

Lemma 3.11. *If Π_{rFE} is 1-NA-SIM-secure, then for all efficient adversaries $\mathcal{A}$, we have that* $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between Hyb_1 and Hyb_2. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that can distinguish between $\mathsf{Real}_{\Pi_{\mathsf{rFE}},\mathcal{A}}(1^\lambda)$ and $\mathsf{Ideal}_{\Pi_{\mathsf{rFE}},\mathcal{A},\mathcal{S}}(1^\lambda)$. Algorithm $\mathcal{B}$ works as follows:

1. At the beginning of the game, algorithm $\mathcal{B}$ is given the public parameters $\mathsf{rFE.pp}$. It constructs the other components of the master secret key msk for the ACE scheme exactly as in Hyb_1 and Hyb_2.
2. Algorithm $\mathcal{B}$ answers the encryption and key-generation queries exactly as in Hyb_1 and Hyb_2. These queries only depend on $\mathsf{rFE.pp}$ (and not $\mathsf{rFE.msk}$ and sank, both of which are unspecified).
3. When $\mathcal{A}$ makes a challenge query $(\mathsf{ct}^*, \mathsf{id}^*)$, algorithm $\mathcal{B}$ queries its decryption oracle on the pair $(F_{\mathsf{Sig.vk},\mathsf{PE.pp}}, \mathsf{ct}^*)$ to obtain a value z^*. It gives z^* to the adversary.
4. At the end of the game, algorithm $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

First, we note that $\mathcal{B}$ does not make any key queries or encryption queries, so it is trivially admissible for the 1-NA-SIM security game. By construction, if the public parameters, the key-generation oracle, the encryption oracle, and the decryption oracle are implemented according to $\mathsf{Real}_{\Pi_{\mathsf{rFE}},\mathcal{A}}(1^\lambda)$, then $\mathcal{B}$ perfectly simulates Hyb_1 for $\mathcal{A}$. In particular, we note that the sanitizer key sank is only needed when responding to the challenge query, and so, the key sampled by the decryption oracle in $\mathsf{Real}_{\Pi_{\mathsf{rFE}},\mathcal{A}}(1^\lambda)$ plays the role of sank. To conclude the proof, we show that if the public parameters, the key-generation oracle, the encryption oracle, and the decryption oracle are implemented according to $\mathsf{Ideal}_{\Pi_{\mathsf{rFE}},\mathcal{A},\mathcal{S}}(1^\lambda)$, then $\mathcal{B}$ perfectly simulates Hyb_2. It suffices to check that the challenge query is correctly simulated.

- In Hyb_2, on a challenge query $(\mathsf{ct}^*, \mathsf{id}^*)$, the challenger computes $y^* \leftarrow \mathcal{S}_4(\mathsf{st}', \mathsf{ct}^*)$. If $y^* = \perp$, then the challenger responds with $\perp$. Otherwise, it parses $y^* = (i^*, \sigma^*, m^*)$, and checks whether $\mathsf{Sig.Verify}(\mathsf{Sig.vk}, i^*, \sigma^*) \stackrel{?}{=} 1$ accepts. If so, it returns $\mathsf{PE.Encrypt}(\mathsf{PE.pp}, i^*, m^*; r)$ where $r \stackrel{\mathrm{R}}{\leftarrow} \mathcal{R}$. Otherwise, it returns $\perp$. This logic precisely corresponds to evaluating $F_{\mathsf{Sig.vk},\mathsf{PE.pp}}(y^*; r)$.
- In the reduction, if the decryption oracle is implemented according to $\mathsf{Ideal}_{\Pi_{\mathsf{rFE}},\mathcal{A},\mathcal{S}}(1^\lambda)$, then the oracle first computes $y^* \leftarrow \mathcal{S}_4(\mathsf{st}', \mathsf{ct}^*)$. If $y^* = \perp$, the oracle returns $\perp$. Otherwise, it returns $F_{\mathsf{Sig.vk},\mathsf{PE.pp}}(y^*; r)$ where $r \stackrel{\mathrm{R}}{\leftarrow} \mathcal{R}$. This is precisely the behavior in Hyb_2.

We conclude that if the oracles are implemented according to $\mathsf{Ideal}_{\Pi_{\mathsf{rFE}},\mathcal{A},\mathcal{S}}(1^\lambda)$, then $\mathcal{B}$ perfectly simulates Hyb_2 for $\mathcal{A}$. The claim then follows by 1-NA-SIM security of Π_{rFE}. □

Lemma 3.12. *If Π_{Sig} is existentially unforgeable, then for all efficient adversaries $\mathcal{A}$, we have that* $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Hybrids Hyb_2 and Hyb_3 are identical except for the extra abort condition in Hyb_3. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between Hyb_2 and Hyb_3 with non-negligible advantage ε. Then, it must be the case that $\mathcal{A}$ can cause Hyb_3 to abort with probability at least ε (otherwise, the two experiments are identical). We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks the security of Π_{Sig}. Algorithm $\mathcal{B}$ works as follows:

1. At the beginning of the existential unforgeability game, $\mathcal{B}$ is given the verification key $\mathsf{Sig.vk}$. Algorithm $\mathcal{B}$ chooses the parameters for the predicate encryption scheme and the functional encryption scheme as in Hyb_2 and Hyb_3. It constructs the sanitizer key sank and msk as in Hyb_2 and Hyb_3, except it leaves $\mathsf{Sig.sk}$ unspecified in msk.
2. During the query phase, $\mathcal{B}$ answers the encryption queries and the decryption key-generation queries exactly as in Hyb_2 and Hyb_3 (since none of these queries depend on knowledge of $\mathsf{Sig.sk}$). Algorithm $\mathcal{B}$ answers the encryption key-generation and challenge queries as follows:
   - **Encryption key-generation queries:** When $\mathcal{A}$ queries for an encryption key for an identity $i \in \mathcal{I}$, algorithm $\mathcal{B}$ submits i to its signing oracle and receives a signature σ. It gives $(\mathsf{rFE.pp}, i, \sigma)$ to $\mathcal{A}$.
   - **Challenge queries:** When $\mathcal{A}$ makes its challenge query (ct^*, i^*), algorithm $\mathcal{B}$ runs the simulator $y^* \leftarrow \mathcal{S}_4(\mathsf{st}', \mathsf{ct}^*)$. If $y^* = \perp$, then $\mathcal{B}$ replies with $\perp$. Otherwise, it parses $y^* = (i^*, \sigma^*, m^*)$, and submits (i^*, σ^*) as its forgery in the existential unforgeability game.

By construction, $\mathcal{B}$ perfectly simulates Hyb_2 and Hyb_3 for $\mathcal{A}$. Thus, with probability at least ε, algorithm $\mathcal{A}$ is able to produce a ciphertext ct^* that causes Hyb_3 to abort. This corresponds to the case where $\mathcal{A}$ never makes an encryption key-generation query for identity i^*, and yet, σ^* is a valid signature on i^*. Since $\mathcal{B}$ only queries the signing oracle when $\mathcal{A}$ makes an encryption key-generation query, by assumption, $\mathcal{B}$ never queries the signing oracle on the message i^*. In this case, σ^* is a valid forgery for the signature scheme, and $\mathcal{B}$ is able to break the security of the signature scheme with non-negligible advantage ε. □

Lemma 3.13. *If Π_{PE} is secure, then for all efficient adversaries $\mathcal{A}$, we have that* $|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between Hyb_3 and Hyb_4. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that can break the security of the predicate encryption scheme Π_{PE}. Algorithm $\mathcal{B}$ works as follows:

1. At the beginning of the game, $\mathcal{B}$ receives the public parameters $\mathsf{PE.pp}$ from the predicate encryption challenger. It samples $(\mathsf{Sig.vk}, \mathsf{Sig.sk})$, $\mathsf{rFE.pp}$, and sank exactly as in Hyb_3 and Hyb_4. It constructs msk exactly as in Hyb_3 and Hyb_4, except it leaves $\mathsf{PE.msk}$ unspecified.
2. During the query phase, $\mathcal{B}$ answers the encryption queries and the encryption key-generation queries exactly as in Hyb_3 and Hyb_4 (since they do not depend on $\mathsf{PE.msk}$). The decryption key-generation queries and the challenge queries are handled as follows:
   - **Decryption key-generation oracle**: When $\mathcal{A}$ queries for a decryption key for an identity $j \in \mathcal{I}$, algorithm $\mathcal{B}$ submits the function $f_{\pi,j} : \mathcal{I} \to \{0,1\}$ (where $f_{\pi,j}(i) = \pi(i,j)$) to the key-generation oracle for the predicate encryption game, and receives the key $\mathsf{PE.sk}_{f_{\pi,j}}$. It gives $\mathsf{PE.sk}_{f_{\pi,j}}$ to $\mathcal{A}$.

– **Challenge oracle**: When $\mathcal{A}$ makes its challenge query $(\mathsf{ct}^*, \mathsf{id}^*)$, algorithm $\mathcal{B}$ first computes $y^* \leftarrow \mathcal{S}_4(\mathsf{st}', \mathsf{ct}^*)$. If $y^* = \perp$, algorithm $\mathcal{B}$ responds with $\perp$. Otherwise, it parses $y^* = (i^*, \sigma^*, m^*)$ and checks the abort condition. If $\mathcal{B}$ does not abort, then it samples a message $m' \xleftarrow{\text{R}} \mathcal{M}$, and submits the pairs (i^*, m^*), (id^*, m') as its challenge query for the predicate encryption security game. The predicate encryption challenger replies with a challenge ciphertext z which $\mathcal{B}$ sends to $\mathcal{A}$.

First, we argue that $\mathcal{B}$ is admissible for the predicate encryption security game. Since Hyb_3 and Hyb_4 behave identically if $\mathcal{B}$ aborts, it suffices to reason about the case where the experiment does not abort. We analyze each case individually:

– If $y^* = \perp$ or $\mathsf{Sig.Verify}(\mathsf{Sig.vk}, i^*, \sigma^*) \neq 1$, then the challenger responds with $\perp$ in both Hyb_3 and Hyb_4 (as does $\mathcal{B}$).
– If $\mathsf{Sig.Verify}(\mathsf{Sig.vk}, i^*, \sigma^*) = 1$, then $\mathcal{A}$ must have previously queried the encryption key-generation oracle on identity i^* (otherwise, the challenger in Hyb_3 and Hyb_4 would have aborted). Since $\mathcal{A}$ is admissible for the no-write security game, for all identities $j \in \mathcal{I}$ that $\mathcal{A}$ submits to the decryption key-generation oracle, it must be the case that $\pi(i^*, j) = 0$. Similarly, by admissibility of $\mathcal{A}$, it must have submitted its challenge identity id^* to the encryption key-generation oracle prior to making its challenge query. Thus, we also have that $\pi(\mathsf{id}^*, j) = 0$. This means that each function $f_{\pi,j}$, that $\mathcal{B}$ submits to the predicate encryption challenger satisfies $f_{\pi,j}(i^*) = 0 = f_{\pi,j}(\mathsf{id}^*)$.

We conclude that $\mathcal{B}$ is admissible. Moreover, if $\mathcal{B}$ is interacting according to $\mathsf{Expt}^{\mathsf{PE}}_{\Pi_{\mathsf{PE}},\mathcal{B}}(\lambda, 0)$, then $\mathcal{B}$ perfectly simulates Hyb_3 for $\mathcal{A}$ and if $\mathcal{B}$ is interacting according to $\mathsf{Expt}^{\mathsf{PE}}_{\Pi_{\mathsf{PE}},\mathcal{B}}(\lambda, 1)$, then $\mathcal{B}$ perfectly simulates Hyb_4 for $\mathcal{A}$. The lemma follows. □

Lemma 3.14. *If Π_{Sig} is existentially unforgeable, then for all efficient adversaries $\mathcal{A}$, we have that* $|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Follows by a similar argument as that used in the proof of Lemma 3.12. □

Lemma 3.15. *If Π_{rFE} is 1-NA-SIM-secure, then for all efficient adversaries $\mathcal{A}$, we have that* $|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Follows by a similar argument as that used in the proof of Lemma 3.11. □

Lemma 3.16. *If Π_{Sig} is perfectly correct and Π_{rFE} is correct, then for all efficient adversaries $\mathcal{A}$,* $|\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Follows by a similar argument as that used in the proof of Lemma 3.10. □

Combining Lemmas 3.10 through 3.16, we conclude that the ACE scheme in Construction 3.1 satisfies the no-write rule. □

4 Extensions

In this section, we describe several extensions to access control encryption that follow immediately from our generic ACE construction in Section 3. We present these extensions primarily as ways of extending the schema of access control encryption to provide increased flexibility, rather than as conceptually new properties achieved by our specific construction. Indeed, it is not too difficult to modify the $i\mathcal{O}$-based ACE construction from Damgård et al. [23] to also provide these properties.

4.1 Dynamic Policies

The access control encryption schema in Section 2.2 required that the access control policies be specified at setup time. In this section, we show how to modify Construction 3.1 so that policies can be associated with individual decryption keys rather than globally. This means that the access control policy no longer has to be fixed at the time of system setup, and moreover, different access control policies can be implemented for each receiver. Thus, the system can support new policies as new receivers are added to the system, and in addition, receivers can update their keys (i.e., obtain new keys from the key distributor) when the access control policies change. Notably, with this extension, changes to the access control policy do not require updating or re-issuing the sender keys. More formally, we would make the following two modifications to the schema of ACE scheme from Section 2.2:

- $\mathsf{ACE.Setup}(1^\lambda) \to (\mathsf{sank}, \mathsf{msk})$: On input the security parameter λ, the setup algorithm outputs the sanitizer key sank and the master secret key msk. Notably, the setup algorithm does *not* take the access control policy π as input.
- $\mathsf{ACE.DKGen}(\mathsf{msk}, j, \pi_j) \to \mathsf{dk}_{j,\pi_j}$: On input the master secret key msk, the receiver identity $j \in \mathcal{I}$, and an access control policy $\pi_j \colon \mathcal{I} \to \{0,1\}$ (the access control policy takes in a sender identity $i \in \mathcal{I}$ and outputs a bit), the decryption key-generation algorithm outputs a decryption key $\mathsf{dk}_{j,\pi}$.

The usual notion of access control encryption from Section 2.2 just corresponds to the special case where the receiver-specific policy π_j is simply the global access control policy π (specialized to the particular receiver identity j). The correctness and security notions generalize accordingly.

Supporting dynamic policies. It is easy to modify Construction 3.1 to support dynamic policies according to the above schema. In fact, policy enforcement in Construction 3.1 is already handled by embedding the access control policy within the receiver's decryption keys. Thus, supporting receiver-specific policies $\pi_j \colon \mathcal{I} \to \{0,1\}$ in $\mathsf{ACE.DKGen}$ can be implemented by simply generating the decryption key as $\mathsf{dk}_{j,\pi_j} \leftarrow \mathsf{PE.KeyGen}(\mathsf{PE.msk}, \pi_j)$. The correctness and security analysis remain unchanged.

4.2 Fine-Grained Sender Policies

As noted in Section 1.1, it is often desirable to support fine-grained sender policies that depend not only on the sender's identity, but also on the contents of the sender's message. In this section, we describe how to extend Construction 3.1 to support fine-grained sender policies. We also give a new security definition (Definition 4.1) to capture the property that a sender should only be able produce encryptions of messages that conform to its particular policy.

Schema changes. In the context of access control encryption, fine-grained sender policies can be captured by modifying the schema for the encryption key-generation algorithm to additionally take in a sender policy (which can be represented as a predicate on the message space of the encryption scheme). Formally, we write

- $\mathsf{ACE.EKGen}(\mathsf{msk}, i, \tau) \to \mathsf{ek}_{i,\tau}$: On input the master secret key msk, a sender identity $i \in \mathcal{I}$, and a sender policy $\tau \colon \mathcal{M} \to \{0,1\}$, the encryption key-generation algorithm outputs an encryption key $\mathsf{ek}_{i,\tau}$.

To support fine-grained sender policies, we first relax the correctness definition (Definition 2.4) by requiring that correctness only holds for messages $m \in \mathcal{M}$ that satisfy the sender's encryption policy. The no-read and no-write rules remain largely unchanged (they are defined with respect to the "always-accept" sender policy). To capture the property that a sender should only be able to encrypt messages for which it is authorized, we introduce a new "soundness" requirement that effectively states that a sender with encryption keys for some collection of policies $\tau_1, \ldots, \tau_Q$ cannot produce a new ciphertext ct that encrypts a message m (with respect to some decryption key dk) where $\tau_k(m) = 0$ for all $k \in [Q]$. More formally, we define the following soundness property:

Definition 4.1 (Soundness). *Let $\Pi_{\mathsf{ACE}} = (\mathsf{ACE.Setup}, \mathsf{ACE.EKGen}, \mathsf{ACE.DKGen}, \mathsf{ACE.Encrypt}, \mathsf{ACE.Sanitize}, \mathsf{ACE.Decrypt})$ be an ACE scheme over an identity space $\mathcal{I}$ and a message space $\mathcal{M}$. Let $\mathcal{A}$ be an efficient adversary and $\pi \colon \mathcal{I} \times \mathcal{I} \to \{0,1\}$ be an access control policy. For a security parameter λ, we define the soundness experiment $\mathsf{Expt}^{(\mathsf{Sound})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda)$ as follows. The challenger begins by sampling $(\mathsf{sank}, \mathsf{msk}) \leftarrow \mathsf{ACE.Setup}(1^\lambda, \pi)$. The adversary $\mathcal{A}$ is then given access to the following oracles:*

- ***Encryption oracle.*** *On input a message $m \in \mathcal{M}$, and a sender identity $i \in \mathcal{I}$, the challenger first generates a sender key $\mathsf{ek}_i \leftarrow \mathsf{ACE.EKGen}(\mathsf{msk}, i, \tau)$, where $\tau(m) = 1$ for all $m \in \mathcal{M}$. The challenger responds with the ciphertext $\mathsf{ct} \leftarrow \mathsf{ACE.Sanitize}(\mathsf{sank}, \mathsf{ACE.Encrypt}(\mathsf{ek}_i, m))$.*
- ***Encryption key-generation oracle.*** *On input a sender identity $i \in \mathcal{I}$ and a sender policy $\tau \colon \mathcal{M} \to \{0,1\}$, the challenger responds with an encryption key $\mathsf{ek}_{i,\tau} \leftarrow \mathsf{ACE.EKGen}(\mathsf{msk}, i, \tau)$.*
- ***Decryption key-generation oracle.*** *On input a receiver identity $j \in \mathcal{I}$, the challenger responds with a decryption key $\mathsf{dk}_j \leftarrow \mathsf{ACE.DKGen}(\mathsf{msk}, j)$.*

At the end of the experiment, adversary $\mathcal{A}$ outputs a ciphertext $\mathsf{ct}^* \in \{0,1\}^*$, and a receiver identity $j^* \in \mathcal{I}$. The output of the experiment is 1 if and only if the following conditions hold:

- $\mathsf{ACE.Decrypt}(\mathsf{ACE.DKGen}(\mathsf{msk}, j^*), \mathsf{ACE.Sanitize}(\mathsf{sank}, \mathsf{ct}^*)) = m^*$ *for some* $m^* \in \mathcal{M}$.
- *Let* $\{(i_k, \tau_k)\}_{k \in [Q]}$ *be the queries* $\mathcal{A}$ *makes to the sender key-generation oracle. For all* $k \in [Q]$ *where* $\pi(i_k, j^*) = 1$, $\tau_k(m^*) = 0$, *where* m^* *is the decrypted message defined above.*

We say that Π_{ACE} *is sound if for all policies* $\pi \colon \mathcal{I} \times \mathcal{I} \to \{0,1\}$, *and all efficient adversaries* $\mathcal{A}$,

$$\Pr\left[\mathsf{Expt}^{(\mathsf{Sound})}_{\Pi_{\mathsf{ACE}},\mathcal{A},\pi}(\lambda) = 1\right] = \mathsf{negl}(\lambda).$$

Supporting sender policies. It is straightforward to extend Construction 3.1 to support *arbitrary* sender policies with little additional overhead. Concretely, we make the following changes to Construction 3.1:

- Instead of a signature on the identity i, the encryption key for an identity $i \in \mathcal{I}$ and sender policy $\tau \colon \mathcal{M} \to \{0,1\}$ contains a signature on the tuple (i, τ), as well as a description of the policy. Namely, $\mathsf{ek}_i = (\mathsf{rFE.pp}, i, \tau, \sigma)$ where $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, (i, \tau))$.
- An encryption of a message $m \in \mathcal{M}$ under the encryption key $\mathsf{ek}_i = (\mathsf{rFE.pp}, i, \tau, \sigma)$ is an encryption of the tuple (i, τ, σ, m) using Π_{rFE}.
- The (randomized) sanitizer function $F_{\mathsf{Sig.vk},\mathsf{PE.pp}}$ now takes as input the tuple (i, τ, σ, m) and outputs $\mathsf{PE.Encrypt}(\mathsf{PE.pp}, i, m)$ if $\mathsf{Sig.Verify}(\mathsf{Sig.vk}, (i, \tau), \sigma) = 1$ and $\tau(m) = 1$. Otherwise, $F_{\mathsf{Sig.vk},\mathsf{PE.pp}}$ outputs $\perp$. The sanitizer key sank is then a decryption key $\mathsf{rFE.sk}_F$ for the modified sanitizer function: $\mathsf{rFE.sk}_F \leftarrow \mathsf{rFE.KeyGen}(\mathsf{msk}, F_{\mathsf{Sig.vk},\mathsf{PE.pp}})$.

At a high level, the sanitizer key implicitly checks that a sender's message is compliant with the associated policy, and outputs a ciphertext that can be decrypted only if this is the case. Here, the signature is essential in ensuring that the sender is only able to send messages that comply with one of its sending policies. In particular, we show the following theorem. We give the proof in the full version of this paper [38].

Theorem 4.2. *Suppose* Π_{Sig} *is existentially unforgeable and* Π_{rFE} *is a* 1-*NA-SIM-secure functional encryption scheme for randomized functionalities (Definition 2.3). Then the access control encryption scheme from Construction 3.1 with the above modifications satisfies soundness (Definition 4.1).*

Relation to constrained PRFs and constrained signatures. This notion of constraining the encryption key to only produce valid encryptions on messages that satisfy the predicate is very similar to the concept of constrained pseudorandom functions (PRF) [18,19,37] and constrained signatures [41,19,10]. Constrained PRFs (resp., constrained signatures) allow the holder of the secret

key to issue a *constrained* key for a predicate that only allows PRF evaluation on inputs (resp., signing messages) that satisfy the predicate. When extending ACE to support fine-grained sender policies, the encryption key-generation algorithm can be viewed as giving out a constrained version of the corresponding sender key. Our technique for constraining the encryption key by including a signature of the predicate and having the encrypter "prove possession" of the signature is conceptually similar to the technique used in [19] to construct functional signatures and in [10] to construct policy-based signatures. In [19,10], this proof of possession is implemented by having the user provide a non-interactive zero-knowledge proof of knowledge of the signature, while in our setting, it is handled by having the user encrypt the signature under an FE scheme and giving out an FE key (to the sanitizer) that performs the signature verification.

Acknowledgments

We thank Shashank Agrawal and the anonymous reviewers for helpful comments. This work was funded by NSF, DARPA, a grant from ONR, and the Simons Foundation. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

1. S. Agrawal and D. J. Wu. Functional encryption: Deterministic to randomized functions from simple assumptions. In *EUROCRYPT*, 2017.
2. J. Alwen, M. Barbosa, P. Farshim, R. Gennaro, S. D. Gordon, S. Tessaro, and D. A. Wilson. On the relationship between functional encryption, obfuscation, and fully homomorphic encryption. In *Cryptography and Coding*, 2013.
3. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS*, 2005.
4. G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS*, 2005.
5. C. Badertscher, C. Matt, and U. Maurer. Strengthening access control encryption. 2017.
6. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.
7. D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical report, DTIC Document, 1973.
8. M. Bellare and D. Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In *CRYPTO*, 2010.
9. M. Bellare, D. Cash, and R. Miller. Cryptography secure against related-key attacks and tampering. In *ASIACRYPT*, 2011.
10. M. Bellare and G. Fuchsbauer. Policy-based signatures. In *PKC*, 2014.
11. M. Bellare and T. Kohno. A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In *EUROCRYPT*, 2003.
12. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE S&P*, 2007.

13. E. Biham. New types of cryptanalytic attacks using related keys. In *EUROCRYPT*, 1993.
14. D. Boneh. The decision diffie-hellman problem. In *ANTS*, 1998.
15. D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, 2001.
16. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC*, 2011.
17. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, 2007.
18. D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.
19. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.
20. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of sanitizable signatures revisited. In *PKC*, 2009.
21. D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, 2010.
22. C. Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding*, 2001.
23. I. Damgård, H. Haagh, and C. Orlandi. Access control encryption: Enforcing information flow with cryptography. In *TCC*, 2016.
24. D. E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5), 1976.
25. D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7), 1977.
26. Y. Dodis, I. Mironov, and N. Stephens-Davidowitz. Message transmission with reverse firewalls - secure communication on corrupted machines. In *CRYPTO*, 2016.
27. V. Fehr and M. Fischlin. Sanitizable signcryption: Sanitization over encrypted data (full version). *IACR Cryptology ePrint Archive*, 2015, 2015.
28. G. Fuchsbauer, R. Gay, L. Kowalczyk, and C. Orlandi. Access control encryption for equality, comparison, and more. In *PKC*, 2017.
29. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.
30. S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, 2013.
31. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, 2012.
32. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from LWE. In *CRYPTO*, 2015.
33. V. Goyal, A. Jain, V. Koppula, and A. Sahai. Functional encryption for randomized functionalities. In *TCC*, 2015.
34. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, 2006.
35. N. J. Hopper, J. Langford, and L. von Ahn. Provably secure steganography. In *CRYPTO*, 2002.
36. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, 2008.
37. A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS*, 2013.

38. S. Kim and D. J. Wu. Access control encryption for general policies from standard assumptions. *IACR Cryptology ePrint Archive*, 2017:467, 2017.
39. I. Komargodski, G. Segev, and E. Yogev. Functional encryption for randomized functionalities in the private-key setting from minimal assumptions. In *TCC*, 2015.
40. A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, 2010.
41. H. K. Maji, M. Prabhakaran, and M. Rosulek. Attribute-based signatures. In *CT-RSA*, 2011.
42. I. Mironov and N. Stephens-Davidowitz. Cryptographic reverse firewalls. In *EUROCRYPT*, 2015.
43. T. Okamoto and K. Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, 2009.
44. A. O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010, 2010.
45. S. L. Osborn, R. S. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.*, 3(2), 2000.
46. C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO*, 1991.
47. M. Raykova, H. Zhao, and S. M. Bellovin. Privacy enhanced access control for outsourced data sharing. In *FC*, 2012.
48. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.
49. R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2), 1978.
50. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1), 2003.
51. A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS*, 2010.
52. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.
53. R. S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11), 1993.
54. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2), 1996.
55. A. Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, 1984.
56. E. Shi, J. Bethencourt, H. T. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE S&P*, 2007.
57. G. Tan, R. Zhang, H. Ma, and Y. Tao. Access control encryption based on LWE. In *APKC@AsiaCCS*, 2017.