Portunus: Re-imagining Access Control in Distributed Systems

Watson Ladd[*,†] *Akamai*
Tanya Verma[*] *Cloudflare*
Marloes Venema *University of Wuppertal*
Armando Faz-Hernández *Cloudflare*
Brendan McMillion[†]
Avani Wildani *Cloudflare*
Nick Sullivan *Cloudflare*

Abstract

TLS termination, which is essential to network and security infrastructure providers, is an extremely latency-sensitive operation that benefits from access to sensitive key material close to the edge. However, increasing regulatory concerns prompt customers to demand sophisticated controls on where their keys may be accessed. While traditional access-control solutions rely on a highly-available centralized process to enforce access, the round-trip latency and decreased fault tolerance make this approach unappealing. Furthermore, the desired level of customer control is at odds with the homogeneity of the distribution process for each key.

To solve this dilemma, we have designed and implemented Portunus, a cryptographic storage and access control system built using a variant of public-key cryptography called attribute-based encryption (ABE). Using Portunus, TLS keys are protected using ABE under a policy chosen by the customer. Each server is issued unique ABE keys based on its attributes, allowing it to decrypt only the TLS keys for which it satisfies the policy. Thus, the encrypted keys can be stored at the edge, with access control enforced passively through ABE. If a server receives a TLS connection but is not authorized to decrypt the necessary TLS key, the request is forwarded directly to the nearest authorized server, further avoiding the need for a centralized coordinator. In comparison, a trivial instantiation of this system using standard public-key cryptography might wrap each TLS key with the key of every authorized data center. This strategy, however, multiplies the storage overhead by the number of data centers. Deployed across Cloudflare's 400+ global data centers, Portunus handles millions of requests per second globally, making it one of the largest deployments of ABE.

1 Introduction

Transport Layer Security (TLS) is a cryptographic protocol widely used to secure communication and protect data integrity between clients, such as browsers, and servers, who host the websites. In a TLS handshake, the server presents a certificate—containing its public key—to the client, and uses the associated private signing key to create a digital signature. This verifies the website's authenticity and creates a secure connection.

Seeking enhanced performance and security, website operators often enlist the services of infrastructure providers like Content Delivery Networks (CDNs). These providers—offering services such as DDoS protection, load balancing, and caching—run on globally distributed data centers to ensure low latency and high performance, and to maintain availability. They also need to be able to inspect the TLS connection between clients, who are the end users of their customer's websites, and their customer's servers. This process of intercepting a TLS connection at an intermediary point in the network is called TLS termination. To handle TLS termination on behalf of their customers, service providers require access to the private signing key for their respective websites.

However, customers utilizing these services have different degrees of comfort concerning the use of their key material across data centers. For example, European customers may stipulate key storage exclusively within the European Union. Another might demand key storage only in data centers secured with bulletproof glass and laser alarm systems. These customers would like providers to control access to their key material based on geographical and security properties. Given that the TLS handshake is in the critical path of establishing a connection to a website, any latency introduced by key access control methods could significantly disrupt service quality. Additionally, for larger infrastructure providers handling millions of TLS terminations per second, minimizing computational overhead from the access control method is essential to scalability.

Unfortunately, traditional access control mechanisms fall short in this endeavor. Centralized methods of access control [41] require edge data centers to communicate with the network's control plane to access specific keys, leading to an expensive round-trip which adds latency and reduces reliability. Alternately, access control using standard public-key encryption provides low latency by assigning

[*] Equal contribution
[†] Work done while at Cloudflare

unique encryption keys to each data center and encrypting the customer's private signing key with the keys of each data center that complies with the access policy. This encrypted data can then be disseminated across all edge data centers in advance of connection requests, reducing latency. However, this strategy becomes rather complex to manage in the face of heterogeneous policies and large scale. The ciphertext size grows in proportion to the number of data centers, creating large overheads. Newly added centers cannot participate in establishing TLS connections unless the customer's signing key is re-encrypted with their newly issued encryption keys.

To address these issues, we required a more direct way to enforce access control through cryptography. Our first attempt [52, 53] combined identity-based encryption [11, 51] and broadcast encryption [25], but ultimately was too inflexible and limited in the types of access policies it could support. Spurred by these restrictions, we created Portunus. Portunus uses a variant of traditional public key cryptography called ciphertext-policy attribute-based encryption (CP-ABE) [9], which can implement fine-grained access control on a cryptographic level. CP-ABE is a a variant of the more general notion of attribute-based encryption (ABE), which was first proposed by Sahai and Waters [46] as a type of public-key encryption in which the keys and ciphertexts are associated with attributes instead of individual users. Concretely, CP-ABE links the secret keys to the attribute set of the key holders, and the ciphertexts to access policies that govern which key holders can decrypt them. Those policies are determined by the encryptor, who can therefore manage access to their data in the spirit of attribute-based access control (ABAC) [38].

We have adopted Portunus at scale. Using Portunus, TLS keys are encrypted using an X25519 key that serves as a data encryption key, which we call the *policy key*. This policy key is further encrypted using ABE under a policy chosen by the customer. Both the encrypted customer keys and policy keys are stored in a globally replicated database present on every machine at Cloudflare. Each edge machine has attributes determined by a database mapping its core cryptographic identity to a set of attributes, e.g., country and region. Edge machines are issued unique ABE secret keys by a key generation authority run in the control plane, allowing them to decrypt only the policy keys that they are authorized to access based on their attributes. Thus, both the encrypted customer keys and policy keys can be stored at the edge, with access control enforced passively through ABE. If a server receives a TLS connection but is not authorized to decrypt the necessary TLS key, the request is forwarded directly to the nearest authorized server, further avoiding the need for a centralized coordinator. As new machines are added, they automatically have access to the keys to which they are permitted by the policy.

While decryption in ABE is more computationally expensive than its equivalent in traditional public key cryptography, we are able to significantly mitigate its impact through session resumption and caching decrypted policy keys.

Adopting CP-ABE as a storage-layer access control solution means that all nodes share the same data, simplifying the distribution process. It also makes it easy for newly added nodes to take up the burden of satisfying requests. Furthermore there are no centralized components whose failure would lead to breaks in the availability of the system. Cryptographically enforced access control is inherently less coupled and more fault tolerant than a centralized system would be.

Our core contributions are:

1. Portunus, a real-world deployment of an ABE-based access control system for key management. Although several works have shown interest in using ABE [22, 23, 32, 48], few have resulted in large-scale real-world deployments.

2. A discussion of the practical costs and benefits of such a scheme, concluding that it is effective in solving distributed access control

3. Lessons learned for future use of CP-ABE by engineers and for ABE researchers about real-world requirements

2 Requirements

In the design process for Portunus, we identified a series of requirements arising from customer needs, internal engineering demands, and the experience of operating the predecessor of Portunus, Geo Key Manager [52, 53].

Low computational overhead: As TLS handshakes can happen at extremely high volumes for legitimate reasons, it is essential that we not add significant computational overhead to the responding process.

Rotation capable: It should be easy to rotate encryption keys used in the system. Key rotation is the practice of systematically replacing cryptographic keys with new ones periodically, to limit the amount of data exposed by the compromise of a particular key. A rotation ensures that newly-uploaded TLS signing keys are not decryptable by machines that have not been updated with new key material.

Recovery from strong attackers: We assume an attacker that is capable of compromising multiple edge machines and reading the database of certificates and associated signing keys. We would like this attacker to be unable to continue impersonating sites after their access is removed, unless the site's certificate was decryptable on the machines they compromised. We also want subsequent certificates not to be decryptable by the attacker after key rotation.

Flexible attributes: Historically, the set of attributes customers want changes over time, such as when a new compliance standard is introduced. Accommodating these changes in the prior system, Geo Key Manager, took considerable work.

Flexible policies: From experience, we know that customers and internal services would need a wide range of policies. Even

if the eventual product did not expose the full expressiveness, future developments would be difficult to anticipate.

Limited storage: Quicksilver, Cloudflare's configuration management system [43], has limited space because it duplicates all data across all machines globally. To preserve fault tolerance and the ability to serve requests quickly from all machines, our system needs to minimize storage overhead.

Uniformity of data: Quicksilver employs a homogeneous tree replication strategy: data centers around the world are organized into a tree and writes at the root are replicated downward. As a response to server failure, the tree is reorganized: such reorganization requires all nodes in the tree to be accessing the new data. Therefore, the system must accommodate a consistent data view across all edge machines.

3 Cryptographic Building Blocks

This section describes the various components of the ABE scheme implemented in Portunus. We start by describing the language to specify policies and attributes. Next, we define CP-ABE and its security property, collusion resistance. After that, we delve into pairings, a mathematical operation used to build many ABE schemes. This includes the scheme of our choice, TKN20, which uniquely satisfies all of our requirements. Presented informally and intuitively, we strive to make this complex scheme accessible to a broad audience. We further discuss necessary aspects for achieving strong security guarantees. Finally, we conclude by discussing the software implementation of this scheme and a usage example of the ABE library API.

3.1 Policy Specification Language

In Portunus, the set of attributes assigned to data centers is an injective map from labels to values, both represented as strings, e.g., `country: Japan`. The policies that are enforced on the wrapped private keys are non-monotone Boolean formulas (consisting of AND, OR and NOT operators) over statements that demand that a label has a value, or that it does not have a certain value, e.g., `country: Japan` or `country: not Japan`. Table 1 shows some example policies and corresponding semantics.

For the negations (i.e., NOT operators), we put the NOT operator on the attribute value rather than on the entire attribute. This means that, to satisfy a negation, e.g., `country: not Japan`, the attribute set must have an attribute with the same label, i.e., `country`, and it must differ from the value i.e., `Japan`. In contrast, many schemes put the NOT on the entire attribute, e.g., `not country: Japan` [40]. In these schemes, the attribute set satisfies the negation if it does not contain the attribute `country: Japan`. However, the problem with this type of negation is that attribute sets that do not have any attributes with this label trivially satisfy this negation. This is especially problematic when new labels are added. Then, all previously issued keys automatically satisfy the negation, regardless of whether they may have the negated value or not.

To express and represent policies, we implement a simple language that parses strings from the API and converts them into the structures that are consumed by the ABE scheme (Section 3.2.7). This means the front end of our policy language is composed of Boolean expressions as strings, such as `country: JP or (not region: EU)`, while the back end is a monotonic Boolean circuit consisting of wires and gates.

Monotonic Boolean circuits only include AND and OR gates. In order to handle NOT gates, we assign positive or negative values to the wires. Every NOT gate can be placed directly on a wire because of De Morgan's Law, which allows the conversion of a formula like `not (X and Y)` into `not X or not Y`, and similarly for disjunction.

3.2 Attribute-Based Encryption

Attribute-based encryption (ABE) is a variant of public-key cryptography in which the key pairs are associated with attributes rather than individual users [46]. Unlike traditional public-key encryption, ABE allows users to enforce a more fine-grained access control to the encrypted data [3, 9, 27, 42, 59]. There are two variants of ABE: key-policy ABE (KP-ABE) [27], and ciphertext-policy ABE (CP-ABE) [9].

3.2.1 Key-Policy ABE (KP-ABE)

In KP-ABE, users' secret keys are generated based on an access policy that defines the privileges scope of the concerned user, and data are encrypted over a set of attributes. For example, consider a military setting. A confidential document about nukes is encrypted under the attributes `type: nuclear, clearance: top-secret`. Then a user with a key defined over the access policy `(type: nuclear or type: laser) and clearance: top-secret` can decrypt the document, but a user with a key `clearance: top-secret` cannot.

3.2.2 Ciphertext-Policy ABE (CP-ABE)

In CP-ABE, encrypting users specify access policies that determine who is allowed to decrypt the data. Users' secret keys are generated over a set of attributes. For example, consider a hospital setting in which a doctor has attributes `role: doctor` and `region: US`, while a nurse has attributes `role: nurse` and `region: EU`. A document encrypted under the policy `role: doctor or region: EU` can be decrypted by both the doctor and nurse.

We restrict our discussion to CP-ABE in this paper, because it is a more natural fit to the desired semantics of Portunus: our fleet of servers have natural attributes like location and compliance standards, and our customers choose their policies.

Table 1: Example Policies and Semantics

Example Policy	Semantics
`country: US or region: EU`	Decrypt only in US or European Union
`NOT (country: RU or country: US)`	Do not decrypt in Russia and US
`country: US and security: high`	Decrypt only in US data centers with a high level of security

3.2.3 Formal Definition of CP-ABE

A ciphertext-policy ABE (CP-ABE) scheme consists of four algorithms [9]:

- Setup$(\lambda) \rightarrow$ (MPK, MSK): The setup takes as input a security parameter λ, it outputs the master public-secret key pair (MPK,MSK).
- KeyGen$(\text{MSK}, S) \rightarrow \text{SK}_S$: The key generation takes as input a set of attributes S and the master secret key MSK, and outputs a secret key SK_S.
- Encrypt$(\text{MPK}, \mathbb{A}, M) \rightarrow \text{CT}_{\mathbb{A}}$: The encryption takes as input a plaintext message M, an access policy $\mathbb{A}$ and the master public key MPK. It outputs a ciphertext $\text{CT}_{\mathbb{A}}$.
- Decrypt$(\text{SK}_S, \text{CT}_{\mathbb{A}}) \rightarrow M'$: The decryption takes as input the ciphertext $\text{CT}_{\mathbb{A}}$ that was encrypted under an access policy $\mathbb{A}$, and a secret key SK_S associated with a set of attributes S. It succeeds and outputs the plaintext message M' if S satisfies $\mathbb{A}$. Otherwise, it aborts.

A scheme is called correct if decryption of a ciphertext with secret key yields the original plaintext message.

3.2.4 Collusion Resistance

The security models for ABE schemes consider their security against chosen-plaintext (CPA) and chosen-ciphertext attacks (CCA), as well as their collusion resistance. Informally, collusion resistance ensures that multiple users with secret keys cannot join forces and decrypt a ciphertext that they could not decrypt individually. For example, a ciphertext encrypted under the policy `role: doctor and region: EU` cannot be decrypted by a user with the attributes `role: doctor` and `region: US`, and another user with the attributes `role: nurse` and `region: EU`. To capture this type of security, the security models allow the attacker to request multiple secret keys for attributes that are not authorized to decrypt the challenge ciphertext. Furthermore, the models capture security against chosen-plaintext attacks or chosen-ciphertext attacks. We define these security models more formally in Appendix A.

3.2.5 Pairing-Based ABE

A popular type of ABE is pairing-based ABE, because it is efficient and can support many desirable properties [59]. A pairing—also known as a bilinear map—is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ defined over three groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p with generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ such that (i) $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for all $a, b \in \mathbb{Z}_p$ (bilinearity), (ii) $e(g_1, g_2)$ is not the identity in $\mathbb{G}_T$ (non-degeneracy) and (iii) e is efficiently computable. Note that $\mathbb{Z}_p$ denotes the ring of integers modulo p.

Intuitively, pairings are used to ensure that we can achieve security guarantees for both the keys and the ciphertexts. We need those guarantees, because we require ABE schemes to be secure against collusion, meaning that users should not be able to combine their keys and obtain better decryption powers. In contrast, traditional public-key encryption typically only provides security guarantees for the ciphertexts. Therefore, we can use discrete-log based assumptions such as the Diffie-Hellman assumption [19] to create secure encryption schemes such as the ElGamal encryption scheme [26]. In such encryption schemes, the public key and ciphertext typically live in a group in which the discrete-log problem is believed to be hard, while the associated secret key is an integer. By exponentiating a part of the ciphertext with the secret key, we can obtain the message. To ensure that we can achieve similar security assumptions for the keys in ABE, we also place the keys in a group in which the discrete-log problem is believed to be hard. To recover the message, we perform a pairing operation instead of exponentiating, which can be seen as an exponentiation with a "hidden" integer.

Most ABE implementations rely on open-source libraries for the pairing-based arithmetic, e.g., MIRACL [49], RELIC [4] or our own library, CIRCL [1]. In this way, ABE can be implemented in a highly optimized fashion without requiring all the details about the inner workings of pairings. Furthermore, using pairings in a black-box way also allows us to efficiently update the underlying pairing-friendly curves, should the old ones be broken or more efficient ones be found [18].

3.2.6 The TKN20 Scheme

We are using a fully CCA-secure hybrid encryption scheme based on the scheme by Tomida, Kawahara and Nishimaki (TKN20) [54–56]. We have open-sourced this code as part of our cryptographic library, CIRCL [1]. We chose TKN20 because it is currently the only ABE scheme that has a full description and satisfies the following properties simultaneously [59]:

1. **Expressivity:** support for AND, OR and NOT operators. Many schemes exist that support monotone formulas, i.e., formulas with AND and OR only. Few of these also

support NOT operators[1].

2. **(Almost) completely unbounded:** any string can be used as an attribute, and there are no bounds on the policy lengths and attribute sets. Note, however, that it is bounded in the number of label occurrences in the secret key, i.e., each label may occur only once.

3. **Multi-use of attributes:** support for repeated use of the same attribute in a Boolean formula.

4. **Strong security guarantees:** full security against chosen-plaintext attacks under standard assumptions[2].

3.2.7 Representation of Monotone Access Policies

In the mathematical description of the scheme, the (monotone) access policies are represented as linear secret-sharing scheme (LSSS) matrices [28]. In such matrices, the rows of the matrix are associated with the attributes used in the policy. To determine whether a set of attributes S satisfies the policy, the subset of rows associated with the attributes that also occur in the set can be considered. If the vector $(1,0,...,0)$ is in the span of those rows, then the set satisfies the policy matrix.

More formally, an access policy can be represented as a pair $\mathbb{A} = (\mathbf{A}, \rho)$ such that $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ is an LSSS matrix, where $n_1, n_2 \in \mathbb{N}$, and ρ is a function that maps its rows to attribute values. Then, for some vector with randomly generated entries $\mathbf{v} = (s, v_2, ..., v_{n_2}) \in \mathbb{Z}_p^{n_2}$, the i-th share of secret s generated by this matrix is $\lambda_i = \mathbf{A}_i \mathbf{v}^\intercal = A_{i,1} s + \sum_{j \in \{2,...,n_2\}} A_{i,j} v_j$, where $\mathbf{A}_i$ denotes the i-th row of $\mathbf{A}$. In particular, if S satisfies $\mathbb{A}$, then there exist a set of rows $\Upsilon = \{i \in \{1,...,n_1\} \mid \rho(i) \in S\}$ and coefficients $\varepsilon_i \in \mathbb{Z}_p$ for all $i \in \Upsilon$ such that $\sum_{i \in \Upsilon} \varepsilon_i \mathbf{A}_i = (1,0,...,0)$, and by extension $\sum_{i \in \Upsilon} \varepsilon_i \lambda_i = s$, holds.

An efficient method to convert a Boolean formula to an LSSS-matrix representation was proposed by Lewko and Waters [36]. For example, the policy `role: doctor and region: EU` is represented as $(\mathbf{A}, \rho)$ where $\mathbf{A} = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}$ and ρ maps the first row to `doctor` and the second row to `EU`. The vector $(1,0)$ can only be recovered from both rows, i.e., by adding them. Note that this algorithm yields the same shares of the secret s as the secret-sharing algorithm in the TKN20 paper.

3.2.8 Representing NOTs and Labels

To represent NOT operators and labels in the policy, we define two additional maps, $\overline{\rho}$ and ρ_{lab}. The map $\overline{\rho}: \{1,...,n_1\} \to \{0,1\}$ maps the rows of the matrix (which each correspond to an attribute in the policy) to 0 if the attribute is not negated, and to 1 if the attribute is negated, e.g., `not region: EU`. The map $\rho_{\text{lab}}: \{1,...,n_1\} \to \{0,1\}^*$ maps the rows of the matrix to labels (represented as strings), e.g., `region`.

3.2.9 High-Level Overview of the TKN20 Scheme

Before we give a description of a simplified version of the TKN20 scheme, we first give an overview of the scheme. By doing this, we aim to demystify the many components of the scheme and highlight the techniques used to construct it.

First, we consider the general form of the scheme's master public key, the secret keys and the ciphertexts. In general, the ciphertext consists of one element in $\mathbb{G}_T$ that hides the message, i.e., $M \cdot A^s$, where $A = e(g_1, g_2)^\alpha$ is part of the public key, and further, elements in $\mathbb{G}_1$ and $\mathbb{G}_2$. The secret keys consists of elements in $\mathbb{G}_1$ and $\mathbb{G}_2$, where at least one contains α "in the exponent", e.g., $g_1^{\alpha + rb}$. To decrypt, the appropriate key and ciphertext components need to be paired (with e) to recover $A^s = e(g_1, g_2)^{\alpha s}$, and thus, the message M.

To embed the attribute sets and policies in the secret keys and ciphertexts, we use appropriate representations of these in $\mathbb{G}_1$ and $\mathbb{G}_2$. To represent the policies, we use the shares λ_i generated with the matrix representation in Section 3.2.7. In the scheme, these shares occur as B^{λ_i} in the ciphertext, where B is part of the master public key. To represent the attribute label-value pairs, we use two techniques: the hash-based [28] and the polynomial-based [10] approaches. The hash-based approach simply takes as input the attribute string, e.g., `role: doctor`, and hashes it directly into $\mathbb{G}_1$ or $\mathbb{G}_2$. The polynomial-based approach takes as input the string and first hashes it to an element x in $\mathbb{Z}_p$, and then maps it into $\mathbb{G}_1$ or $\mathbb{G}_2$ with an implicit polynomial, e.g., $B_0 \cdot B_1^x = g_1^{b_0 + x b_1}$. In TKN20, these two approaches are combined: a hash is used to map the attribute-label string directly into the group $\mathbb{G}_1$, and the implicit polynomial is used to map the attribute-value string into the group. More specifically, this combination computes $H_0(\text{lab}) \cdot H_1(\text{lab})^x$, where lab denotes the label, e.g., `role`, and x denotes the representation of the associated value, e.g., `doctor`, in $\mathbb{Z}_p$.

The reason why TKN20 maps the attribute values into the group using the polynomial-based approach is that it can support NOT operators. To support these, TKN20 uses the high-level approach introduced by Ostrovsky et al. [40], which exploits the structure of the polynomial-based map. Roughly, this approach uses the fact that two distinct points on a 1-degree polynomial can be used to reconstruct the polynomial with Lagrange interpolation[3]. More concretely, this means that the secret can be reconstructed if the attribute value in the key does not match the attribute value in the ciphertext, i.e., when they represent two distinct points on the polynomial.

3.2.10 Simplified Description of the TKN20 Scheme

We provide a simplified version of the scheme below, and explain then how the real version of the scheme—which can

[1] NOT operators can be supported in three ways [5]. TKN20 supports the most efficient variant proposed by Okamoto and Takashima [39]. This variant requires that the attribute set uses each label at most once.

[2] We do, however, require the use of the random oracle model [7]

[3] This approach is also used in Shamir's secret sharing scheme [50].

be found in the TKN20 paper [55, 56]—can be constructed from the simplified version.

- Setup(λ) $\rightarrow$ (MPK, MSK): The setup outputs the master public-secret key pair (MPK, MSK), where $H_i\colon \{0,1\}^* \rightarrow \mathbb{G}_1$ with $i \in \{0,1\}$ are two hash functions (modeled as random oracles), MSK $= (\alpha, b)$, and

$$\text{MPK} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, H_0, H_1,$$
$$A = e(g_1, g_2)^{\alpha}, B = g_1^b).$$

- KeyGen(MSK, (S, ψ_{lab})) $\rightarrow \text{SK}_S$: On input a set of attribute values S and the associated labeling map $\psi_{\text{lab}}\colon S \rightarrow \{0,1\}^*$, which maps the attributes in the set S to labels (represented as strings), it outputs the secret key SK_S as

$$\text{SK}_S = (S, K_1 = g_1^{\alpha + rb}, K_2 = g_2^r,$$
$$\{K_{3,\text{att}} = (H_0(\psi_{\text{lab}}(\text{att})) \cdot H_1(\psi_{\text{lab}}(\text{att}))^{x_{\text{att}}})^r\}_{\text{att} \in S}),$$

  where $r \in_R \mathbb{Z}_p$ is a randomly generated element in $\mathbb{Z}_p$ and x_{att} denotes the representation of att in $\mathbb{Z}_p$.

- Encrypt(MPK, $\mathbb{A}, M$) $\rightarrow \text{CT}_{\mathbb{A}}$: On input a plaintext message $M \in \mathbb{G}_T$ and an access policy $\mathbb{A} = (\mathbf{A}, \rho, \rho_{\text{lab}}, \overline{\rho}, \tau)$—where $\tau\colon \{1,\ldots,n_1\} \rightarrow \{1,\ldots,m\}$ is a function that maps each row that is associated with the same label to a different integer in $\{1,\ldots,m\}$, with m being the maximum number of times that a label occurs in the policy—it outputs a ciphertext $\text{CT}_{\mathbb{A}}$ as

$$\text{CT}_{\mathbb{A}} = (\mathbb{A}, C = M \cdot A^s, C_1 = g_2^s, \{C_{2,l} = g_2^{s_l}\}_{l \in \{1,\ldots,m\}},$$
$$\left\{C_{3,j} = B^{\lambda_j} \cdot (H_0(\rho_{\text{lab}}(j)) \cdot H_1(\rho_{\text{lab}}(j))^{x_{\rho(j)}})^{s_{\tau(j)}}\right\}_{j \in \chi_0},$$
$$\left\{C_{3,j} = B^{-\lambda_j} \cdot H_0(\rho_{\text{lab}}(j))^{s_{\tau(j)}},\right.$$
$$\left. C_{4,j} = B^{x_{\rho(j)}\lambda_j} \cdot H_1(\rho_{\text{lab}}(j))^{s_{\tau(j)}}\right\}_{j \in \chi_1}),$$

  where $s, s_1, \ldots, s_m, v_2, \ldots, v_{n_2} \in_R \mathbb{Z}_p$ are randomly generated elements in $\mathbb{Z}_p$, $\lambda_j = A_{j,1}s + \sum_{k \in \{2,\ldots,n_2\}} A_{j,k}v_k$, and $\chi_i = \{j \in \{1,\ldots,n_1\} \mid \overline{\rho}(j) = i\}$ for $i \in \{0,1\}$.

- Decrypt($\text{SK}_S, \text{CT}_{\mathbb{A}}$) $\rightarrow M'$: On input the ciphertext $\text{CT}_{\mathbb{A}}$, and a secret key SK_S, it checks whether S satisfies $\mathbb{A}$. If not, then it aborts. Otherwise, it computes the message by first determining $\Upsilon_0 = \{j \in \chi_0 \mid \rho(j) \in S\}$, $\Upsilon_1 = \{j \in \chi_1 \mid \rho(j) \notin S \wedge \rho_{\text{lab}}(j) \in \psi_{\text{lab}}(S)\}$, $\Upsilon = \Upsilon_0 \cup \Upsilon_1$ and $\{\varepsilon_j\}_{j \in \Upsilon}$ such that $\sum_{j \in \Upsilon} \varepsilon_j \mathbf{A}_j = (1,0,\ldots,0)$, then computing

$$e(g_1, g_2)^{\alpha s} = e(K_1, C_1)$$
$$\cdot \left(\prod_{j \in \Upsilon_0} \left(e(K_{3,\rho(j)}, C_{2,\tau(j)}) / e(C_{3,j}, K_2)\right)\right.$$
$$\left. \cdot \prod_{j \in \Upsilon_1} \left(e(K_{3,\rho(j)}, C_{2,\tau(j)}) / e(C_{3,j}^{y_j} \cdot C_{4,j}, K_2)^{\frac{1}{x_{\rho(j)} - y_j}}\right)\right),$$

  where $y_j = x^{-1}_{\psi_{\text{lab}}(\rho_{\text{lab}}(j))}$. Then, $M = C / e(g_1, g_2)^{\alpha s}$.

3.2.11 Description of the Fully Secure Variant

The structure of the actual TKN20 scheme [55] is much more advanced. This is because the scheme is fully secure under well-studied assumptions, in particular, a variant of the matrix decisional Diffie-Hellman assumption [21]. This assumption is closely related to the decisional Diffie-Hellman assumption [19, 21]. The main technique that is used to achieve this level of security is the dual-system encryption technique [62]. Currently, the most advanced and efficient techniques [16, 35] in this paradigm use matrix structures "in the exponent", e.g., mapping the key component $K_1 = g_1^{\alpha + rb}$ to $g_1^{\mathbf{a} + \mathbf{W}\mathbf{r}}$, where $\mathbf{a}$ and $\mathbf{r}$ are vectors of length 3 and $\mathbf{W}$ is a (3×3)-matrix [35].

3.2.12 Support for Wildcards

To support CCA-security more efficiently than e.g., [64], we use wildcards in the secret keys (as also proposed in the journal version of TKN20, i.e., [56]). A wildcard is represented by an asterisk $*$, e.g., `region: *`, and means that all values for the associated label are accepted, e.g., `region: EU`. In other words, it always matches any occurrence of an attribute with the same label in the policy. The keys for asterisks have the following form:

$$(K_{3,1,\text{att}}, K_{3,2,\text{att}}) = (H_0(\psi_{\text{lab}}(\text{att}))^r, H_1(\psi_{\text{lab}}(\text{att}))^r).$$

Tomida et al. [56] show that the variant of the scheme using wildcards is provably fully secure as well. Note that we use this functionality only to achieve CCA-security, because this functionality seems less intuitive to use for other purposes. In particular, handing out a wildcarded attribute for some label gives the user much power: it always satisfies any occurrence of that specific attribute label in the policy, regardless of what the policy dictates that the user should have.

3.2.13 Key Encapsulation and Symmetric Encryption

We use the TKN20 scheme to encapsulate a symmetric key to be used to encrypt the data, and use a one-time secure symmetric encryption scheme to encapsulate the data. More accurately, we first derive a symmetric key from the ABE ciphertext. In particular, instead of encrypting some message $M \in \mathbb{G}_T$, we directly derive the symmetric key from $e(g_1, g_2)^{\alpha s}$ by applying a key derivation function [17]. Because $e(g_1, g_2)^{\alpha s}$ is indistinguishable from a random element in $\mathbb{G}_T$, the derived key is also indistinguishable from a random key [31, 33]. Then, we use this random key to symmetrically encrypt the data. For this, we use a symmetric encryption scheme that is one-time secure, which means that no attackers can distinguish between the encryptions of any two messages (see Appendix B for a more formal definition). This "hybrid encryption" variant—where we use ABE to encapsulate a key and symmetric encryption to encapsulate the data—is provably secure against chosen-plaintext attacks, see e.g., [34, §A]. To encrypt symmetrically, we use the same approach as Boneh and Katz [12]. We

use a pseudo-random generator to generate a key stream that has the same length as the message, and XOR it to the message. In Portunus, we use an extendable output function to generate a sufficiently-long key stream, i.e., BLAKE2b [45], which is believed to be indistinguishable from pseudo-random generator.

3.2.14 CCA-Security via the BK-Transform

Finally, to achieve CCA-security, we apply the Boneh-Katz transform [12]. With this transform, we combine the hybrid encryption scheme with a message authentication code (MAC) function and a special commitment scheme[4], for which formal definitions and security models can be found in Appendix B. Informally, the special commitment scheme that we use consists of two independent hash functions. The first hash is used to generate a public commitment to a secret random value, and the second hash uses the secret random value to derive a key K'. Subsequently, this secret random value is included in the encryption of the message with the hybrid encryption scheme. We compute a MAC with the key K' over the resulting ciphertext to ensure authenticity of the ciphertext. Furthermore, the public commitment to the secret random value is included in the access policy with an AND operator applied to the original policy, and also in plain in the resulting ciphertext. To decrypt, one first recovers the message and secret random value by decrypting the ciphertext. Then, one verifies whether the public commitment is equal to the hash over the secret random value, and then if the MAC verifies correctly given the key derived from the secret random value. We give a full description of the CCA-secure construction (using the simplified version of TKN20) in Appendix C. It follows from [12] and [60] that this construction is CCA-secure.

3.3 Software Implementation

We implemented our scheme as part of the CIRCL library [1] in Go. The particular instantiation of the pairing-friendly groups $\mathbb{G}_1$,$\mathbb{G}_2$ and $\mathbb{G}_T$ that our implementation uses is the BLS12-381 curve [6, 13]. We generated the code for the arithmetic in $\mathbb{Z}_p$ with the Fiat Cryptography tool [20], which formally verifies the correctness of the produced code. We have also optimized the arithmetic for the groups $\mathbb{G}_1$,$\mathbb{G}_2$ and $\mathbb{G}_T$ through judicious choice of representation. Our implementation uses the fast subgroup checks via Bowe's method [14], which allow us to check whether any given point is in the group, e.g., $\mathbb{G}_1$. To hash into groups, we followed the relevant IETF specification [24]. To optimize the decryption algorithm, we use two common tricks that are often used to speed up computing a product of pairings, i.e., by reordering the computations [42] and by sharing the final step of the pairing operations [29]. Figure 1 presents a reproducible program showing the usage of our code.

[4]Boneh and Katz [12] call this an "encapsulation" scheme, but to distinguish it more clearly from key and data encapsulation, we call it "special commitment scheme" in this paper.

4 Design

Armed with the above scheme, we now must construct services to encrypt customer keys, and make them available to those who should have them. Cloudflare logically has four components. The first is a set of edge machines located in geographically spread and distant data centers. These edge machines run a homogeneous mixture of services that terminate TLS and serve HTTP. The actual signing of TLS handshakes takes place in a system called Gokeyless in all relevant cases.

The second component is a centralized set of services in the control-plane responsible for the API that customers interact with to configure their website. One of these services, the certificate manager, handles all configuration relating to TLS.

The third component is a small number of very tightly controlled machines that handle certificate issuance for internal certificates. All machines at Cloudflare have a machine identity based on RSA keys: our key issuance service uses that identity to determine the attributes a machine shall have. We call this service the Key Generation Authority (KGA).

The fourth component is a globally synchronized key-value store, Quicksilver. This is a global gossip tree for customer configurations, such as certificates, that is designed to ensure extremely fast replication, at the cost of constrained bandwidth and storage. Every edge machine stores a local copy of the data in Quicksilver.

4.1 Encrypting Customer Keys

When a customer uploads a certificate and the associated private signing key to Cloudflare, and indicates it is to be protected under an access policy, the certificate manager in the control-plane takes the private key and encrypts it with the required policy. However, the customer's private key is not encrypted with the ABE master public key directly. Rather, it is encrypted with an X25519 key pair, the private key of which is encrypted under the ABE scheme. These key pairs are indexed by the policy and the epoch they are under. At any time, there may be several of these key pairs, called *policy keys*, present in the database for a given policy. The certificate manager will use the most recent one for encryption. This permits gradual rotation of the key pairs. Note that the only encryption happening in Portunus is done by the certificate manager.

4.2 Accessing Customer Keys

On receipt of a connection to a site, such as `alice.test.com`, Gokeyless carries out a lookup for the certificate in Quicksilver. If that certificate has a key protected by Portunus, the metadata for that certificate will have a pointer to the relevant policy key together with a ciphertext that decrypts to the private key. Gokeyless then loads the policy key and determines if it is decryptable by the machine. If not, it consults a table that maps each policy to a list of satisfying data centers to find a

Listing 1: Example usage of the ABE library API

```
masterPubKey, masterSecKey := Setup() // Initialize the master public and master secret key
accessPolicy := new(Policy)
accessPolicy.FromString("country: US or region: EU") // Create new policy from given string
// Encrypt the secret message using the master public key and policy
encryptedMsg := masterPubKey.Encrypt(accessPolicy, []byte{"long live ABE"})
parisDCAttributes := new(Attributes) // Create attributes for the Paris data center
parisDCAttributes.FromMap(map[string]string{ "country": "FR", "region": "EU"})
// Generate an attribute secret key for the Paris data center using the master secret key
parisDCSecKey := masterSecKey.KeyGen(parisDCAttributes)
// Decrypt the ciphertext using the attribute secret key of the Paris data center
decryptedMsg := parisDCSecKey.Decrypt(encryptedMsg)
assertEquals(decryptedMsg, []byte{"long live ABE"})
```

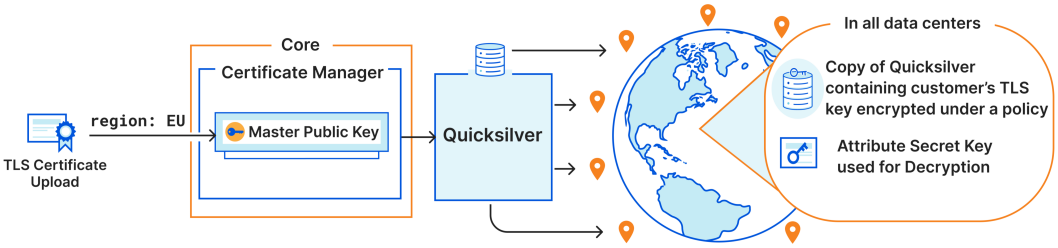

Figure 1: Encryption under a policy

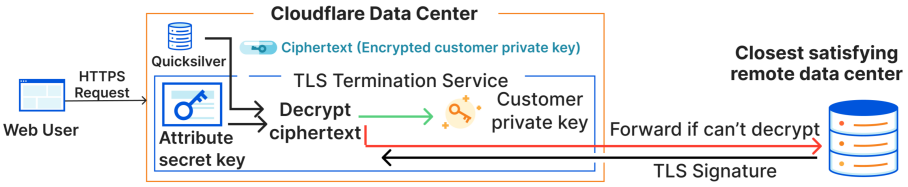

Figure 2: Decryption using Attribute Secret Key

neighboring one, and forwards the request there. Gokeyless on this machine then decrypts the policy key and uses the result to decrypt the certificate's private key, performing the signature and completing the TLS handshake. The decrypted policy keys are cached in memory, so the computationally burdensome ABE decryption only happens once for commonly used policies. This is an important optimization to avoid excessive CPU consumption during attack scenarios when many handshakes are arriving.

4.3 Key Distribution

The key generation authority (KGA) holds the ABE master secret key. It also has access to the unique cryptographic identity for every machine in the fleet, as well as a map of machines to attributes. This map is largely synchronized with the machine's own view. Key issuance for the machine's attribute-based secret key is managed by the service configuration management system, Salt [2]. Salt uses the RSA identity key of the machine to authenticate to the CA, which generates the machine's attribute secret key using the master secret key and the attributes of the machine. The map of machines to attributes is configured in the same database that drives machine identity for Salt.

4.4 Key Rotation

Over time, it is necessary to change the key material in the system so that an attacker who has access to old key material can no longer decrypt newly uploaded customer TLS private keys. However, the lifetime of a customer certificate can extend beyond a rotation period and it must be possible to continue to decrypt the customer TLS key for that duration.

The key generation authority generates a new *generation* of the master key pair. To preserve the ability to decrypt old TLS private keys, the CA re-encrypts the existing policy keys on behalf of the certificate manager. During this process,

machines will have both the old and new generation of the attribute secret key, ensuring that availability is not impacted as the old key material is phased out.

Newly uploaded certificate private keys are encrypted under the same policy key. This means that an attacker who has access to a policy key can continue to to decrypt new TLS keys, but it is possible to generate new policy keys for a policy. This does however guard against an attacker who obtains the attribute secret keys for a machine from being able to access the TLS keys after rotation.

To detect accidental or malicious usage of expired key generations and have end-to-end visibility into the status of key rotation, we have logging and metrics for key generation held and used in each part of the system.

4.5 Attribute Changes

From time to time, the attributes associated with a set of data centers may change. Introducing new labels that have not been used by existing policies is straightforward, since the set of data centers that can decrypt a given TLS private key remains unchanged. However, when the attributes of the data centers that can decrypt a key are changing, certain changes are needed to maintain system functionality [5, 59]. We split the act of "changing an attribute" in two steps: removal of the old label and value, and the re-addition of the label with the new value. The former results in loss of decryption capabilities for associated data centers, because they no longer satisfy the policies that required the presence of this label. Note that the removal does not increase the decryption capabilities yet, because to satisfy a negated attribute, the set of attributes of the data center must have an attribute with the same label, regardless of the value. Adding a new label can only increase the number of policies satisfied due to negation semantics.

Carrying out this transition requires three steps. First, the affected label is removed from the forwarding information of the involved data center, so that other data centers stop sending requests that require its presence. Second, the key is re-issued with the new attribute. Third, the new attribute is re-added to the forwarding information so requests are handled by the data center again. Throughout this, the affected data center handles end-user requests as usual: those requests that cannot be satisfied locally are forwarded to other data centers that can satisfy them, whose forwarding information is not affected by the transition. This process can be difficult to carry out at scale and requires careful planning and should be done in stages. Lastly, a key rotation is required to ensure that any retained copies of the older key are not used.

4.6 Networking and Resiliency

Gokeyless uses an RPC protocol to forward TLS signing requests to the closest satisfying data center, which on arrival leverage a network layer load balancer [63] to determine an appropriate machine to handle the connection. Since the computational load of handling a request forwarded for Portunus is merely an X25519 decryption and an RSA/ECDSA signature, even high levels of request volume have not led to failures due to load balancing issues.

Because maintaining connections to all other machines can be expensive, machines within one data center will elect among themselves a machine to forward requests to specific close foreign data centers. This reduces the number of TCP connections being used.

Resiliency is negatively impacted for customers who apply overly restrictive policies. It is possible for data centers to be taken offline or become overwhelmed for a variety of reasons. If all data centers a customer's key is decryptable in are offline, then the customer's website will be rendered inaccessible. To prevent this, we require customer keys be decryptable in at least two large-capacity data centers.

We have found that typically customers will store their certificates in regions where the majority of their users are found. This unsurprising pattern puts low demands on the remote execution capabilities. Unfortunately, events such as DDoS attacks can add significant load. Operation under normal conditions is not a guide to operation under adverse conditions. This led us to expend significant effort to integrate distributed tracing in addition to metrics to track system performance and quickly diagnose and reproduce scaling issues.

5 Evaluation

While Portunus was launched to customers in 2022, the older version of the system based on similar principles (Geo Key Manager) has been in production since 2017. Over the years, the number of customers and end-users relying on this product has steadily increased. This section is an evaluation of the various components of Portunus.

During a sample week in December 2022, we observed 100k requests per second being served between Portunus and Geo Key Manager. As most customers restrict key access to the region where they typically have the most users, approximately 80% of these requests are handled locally. The remaining 20% are forwarded to their closest satisfying neighbor.

5.1 Cryptography

We evaluate our underlying cryptography library against RSA-2048 and X25519, utilizing Go libraries `crypto/rsa` and `x/crypto/nacl/box` as reference implementations. These comparative algorithms were chosen because they are standard public-key cryptography. We conduct our measurements on an Apple M1 Mac.

We characterize our library's performance using measures inspired by ECRYPT [8]. In all comparisons involving ABE, we set the attribute set size to 50 and consider policy formulas over 50 attributes. This attribute set size is significantly higher

Table 2: Space Overheads (bytes)

Scheme	Secret key[5]	Public key	Encrypt 23 B	Encrypt 10 KB
RSA-2048	1190	256	233	496
X25519	32	32	48	48
Our scheme	23546	3282	19475	19475

Table 3: Operation times (ms)

Scheme	Key Gen.	Encrypt 23 B	Decrypt 23 B
RSA-2048	180	0.209	1.47
X25519	0.061	0.096	0.046
Our scheme	701	364	30.1

than necessary for any of Portunus' applications, as most policies are typically limited to a combination of geographic properties. Nevertheless, it serves as an extreme worst-case scenario for benchmarking purposes.

Table 2 shows the space consumed by the various operations. For our system, the ciphertext overhead is of particular concern since it is replicated on every machine. Unfortunately, this overhead is significantly larger than in traditional public-key cryptography. However, the good news is that this overhead is constant with respect to message length for a given policy size, and can be reduced by relying on a small handful of policy keys (defined in Section 4.1) rather than encrypting every customer key using ABE. Importantly, the ciphertexts in our system can be decrypted by multiple decryptors, whereas standard public-key cryptography benchmarks only consider a ciphertext that can be decrypted by a single decryptor. The size of the attribute secret key is less relevant, as a single copy is stored per machine. The size of the master public key is of even less concern, as it is only used by the certificate manager in the control plane.

Table 3 shows the average time required to perform different key operations. Key generation refers to the process of generating attribute secret keys from the master secret key, which can be performed out-of-band of user handshakes and is therefore of marginal relevance in this context. Encryption latency can also largely be ignored, as it is acceptable for encryption to take a few extra cycles before a certificate is considered deployed. But once it is deployed, HTTPS requests to the website should complete quickly. Since decryption is in the critical path of every request, it is the most pertinent in our situation. While session resumption and caching policy keys can amortize the number of ABE decryptions across TLS handshakes to a small fraction, improvements to decryption latency will still affect overall baseline performance. It is therefore important to further optimize the decryption process.

[5]For ABE, this is the Attribute-Based Secret Key.

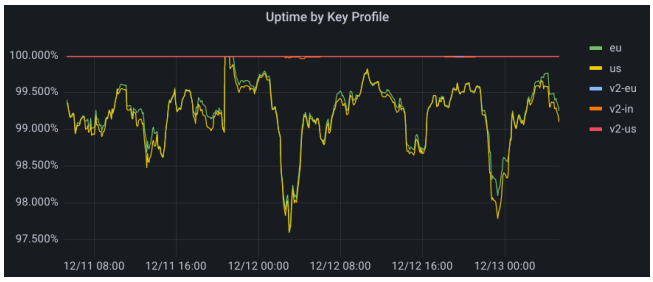

Figure 3: Uptime by policy; this shows that Portunus (v2) has consistently better uptime than Geo Key Manager (v1)

5.1.1 Request Latency

The overall performance of Portunus includes the impact of cryptography, networking and geographic location based on the type of Portunus request: handshakes processed locally, and those forwarded to a remote data center. The vast majority of local requests only perform an X25519 decryption because of policy keys. The remainder incur the overhead of an ABE decryption. For remote requests, network latency largely dominates.

5.2 Availability

Figure 3 shows the uptime of our system by policy vs the previous system. This graph was produced using synthetic probes spanning every machine across our fleet. It demonstrates that dynamically selecting all possible machines to decrypt rather than a pre-determined handful as in our previous release, produces significant improvements to real-world reliability.

6 Discussion

We want to reiterate why an access-control solution based on novel cryptography makes the most sense for our system.

The TLS key management system is responsible for ensuring that edge machines can access customer signing keys when customers upload their TLS certificates (and associated signing keys) to Cloudflare. To ensure availability and low latency, the key manager relies on an internal globally synchronized key-value store, Quicksilver [43] to distribute user configurations to all edge data centers within seconds. A copy of the entire Quicksilver data set is replicated on every edge machine for fault tolerance, allowing requests to be served even if disconnected from the central synchronizing server.

However, augmenting the key management system to support policy-based access restrictions required us to rethink our approach of storing the same data on all edge machines. Using a central server to enforce access would reduce fault tolerance and add additional latency, undermining the advantages of a distributed edge. Alternatively, we could ensure that only data centers that satisfy a given policy receive those policy-restricted keys. However, this would require

modifying Quicksilver's replication strategy to store only a subset of the key set, which challenges core design decisions Cloudflare has made over the years that assume the entire data set is replicated on every machine.

We considered a third option of issuing unique keys for each data center: wrapping each TLS signing key with the key of every authorized data center and adding them all to Quicksilver. Although this approach would have permitted access to the key only in certain locations while letting TLS be terminated where possible, it would also significantly increase the storage space requirements on every machine proportional to the number of datacenters.

This encouraged us to explore alternative cryptographic solutions. Our first attempt, Geo Key Manager, was developed back in 2016, when there was only one ABE scheme that supported all properties [59], but in a rather inefficient way. In particular, at the time, supporting negations led to significant efficiency penalties in the decryption algorithm [58]. Only recently, the community started addressing these efficiency issues [5, 55, 58]. To get around this limitation, we initially used a combination of identity-based encryption and broadcast encryption to simulate an ABE-like scheme. Unfortunately, this scheme was not collusion resistant (Section 3.2.4). As a result we were eager to switch to a more theoretically satisfying solution once practical ABE schemes became available.

During the implemention of our ABE scheme, while we performed much optimization (Section 3.3), the implementation has larger overheads compared to the mature implementations for traditional cryptographic schemes. We mitigated some of these costs by using policy keys (Section 4.1). This approach is similar to hybrid encryption, where public-key cryptography is used to establish a shared symmetric key used to encrypt data. Policy keys are public key encryption keys to permit the central service to encrypt user's certificates without access to the key. While not as efficient as symmetric cryptography this still reduces the overhead.

The performance and reliability improvements of the deployed system are due to side-effects of the new cryptographic scheme. The original system only supported one kind of attribute, a region. Unfortunately, this did not support customers who wanted to specify countries, and so a hard-coded list of cities was used on upload. This list was rarely updated, so new datacenters were not used. Migration was an extremely difficult prospect. Switching to the new system meant that country could be used directly, and additional attributes could be added. This immediately increased the available set of data centers for many common policies, and directly improved reliability.

Although ABE employs a highly trusted key generation authority to issue the secret keys, we argue that this authority does not need to be more trusted than an authority enforcing traditional access control. Specifically, in Portunus, the role of the key generation authority is integrated with a certificate authority that is used to secure all critical services within Cloudflare's internal network. If this authority were to be corrupted, the consequences would be much worse than simply breaking the security of ABE. If, however, a similar setting would require that the trust in the authority is mitigated by distributing the trust across multiple authorities, one could also deploy multi-authority ABE [15, 37].

A policy conundrum may also arise in certain situations: the encrypted data still resides in restricted regions. This can potentially cause concern among those without a comprehensive understanding of the system. Assuaging these concerns will vary between organizations, but a big part involves spreading awareness of how policy-based encryption works.

7 Future Work

Although ABE can support all properties required by our particular application, it does present minor limitations that may be critical in other contexts. For instance, our scheme doesn't support policies with wildcards of the form `country: *`, meaning any country can satisfy this policy. It likewise doesn't permit an attribute set with multiple values for a single attribute label, such as `{group: fiddlers, group: percussionists}`.

It is unclear what post-quantum ABE schemes will have the combination of performance, implementation simplicity, and expressivity required. Likewise, the use of pairing-based cryptography creates some challenges in acceptance, as decision makers may be unfamiliar with it and it is not standardized, despite ongoing efforts towards standardization at IETF [47].

8 Lessons Learned

In the course of operating Portunus and its predecessor Geo Key Manager, we have learnt several lessons.

Even after more practical ABE schemes became available, the difficulty in translating a scheme from an ABE paper to practice, as well as in selecting an appropriate scheme, should not be underestimated. Typically, there are some parameters that must be chosen, with little indication of the strength of the various assumptions the parameters create. In addition, the notation can require a formidable amount of translation, sometimes concealing significant computational steps.

There persists a prevalent notion in the cryptography community that ABE is still unreasonably slow to be useful. We believe this is no longer true. Just like traditional public key cryptography is not used independently to encrypt large amounts of data, but rather in concert with symmetric encryption, we believe many applications can enjoy the benefits of ABE using it with a hybrid encryption strategy.

Complicated cryptographic schemes in services, particularly ones as critical as TLS termination, can elicit operational apprehensions amongst SREs and other teams that depend on the service. Cryptography can end up being scapegoated when issues arise, despite the problems originating from other system components. We believe when designing such a system, it is prudent

to prioritize simplicity in every other aspect. This makes failures outside the cryptography straightforward to diagnose and conserves the complexity budget for the cryptography.

We have certainly faced the consequences of not adhering to this principle, in the form of delayed rollout due to difficulty garnering operational confidence. Geo Key Manager's complexity extended beyond just the cryptographic components, such as the use of a custom RPC protocol - an artifact of the parent system's (which Geo Key Manager was integrated into) development before the existence of gRPC. This custom protocol, despite resolving most of its quirks as part of Portunus's development, continued to present challenges in specific edge cases. We are presently transitioning to gRPC to alleviate these issues and preempt future related issues. Another example of reducing complexity was replacing a complex thread-pool architecture based on outdated assumptions, with a simple and scalable architecture of one goroutine per request, capitalizing on the lightweight concurrency model offered by Go. Our ongoing efforts aim to simplify various other components, with the ultimate goal of improving system maintainability and fostering enthusiastic buy-in from other stakeholders, as well as encouraging other teams to consider applying ABE-based access control for their own use cases, armed with the reassurance of Portunus's successful deployment.

9 Related Work

Prior work has considered CP-ABE for enforcing access control. Oftentimes, the design of new ABE schemes is justified by their new capabilities based on use in access control [9,44,65], but without many details on potential system design. However, as Venema and Alpár demonstrate [57], there have been various attacks on some of these constructions [44,65], particularly those that do not rely on pairings [30,66]. It is therefore important that care is taken in choosing an established scheme that has the necessary properties and is secure.

Although there are many schemes with various properties [59], our chosen scheme is unique in satisfying our desired properties 3.2.6, making comparisons with alternative ABE schemes difficult. Likewise, comparing cryptographic operation benchmarks with potentially sub-optimized research implementations of ABE schemes is not in scope and we refer the reader to more comprehensive analyses [18].

In contrast to most works about systems that use ABE, Sieve [61] dives deeper into the details of the system. The authors discuss how key management, ABE overhead, and key deletion have to be addressed in a deployed system. In this system, applications can interact with user data stored in the cloud, while users maintain control over which applications have access to their data. Because Sieve is built for a different setting than Portunus, it makes different choices in how ABE is applied. Most notably, it uses key-policy ABE, where the keys are associated with policies and the ciphertexts are associated with attribute sets, to apply a "tag-based" access control like described in Section 3.2.1. In particular, the encrypted data objects that are stored in the cloud are associated with attribute sets. Once an application requests access to the user's data, the user can specify a policy stating which data can be accessed.

Excalibur [48] is perhaps the most similar to our work, because it also uses CP-ABE to enforce access control in the spirit of attribute-based access control. The authors designed and implemented a system for customers to make data accessible on certain machines, and integrated it into a cloud environment. Because their use case is different, the challenges that their system overcomes also differ. Importantly, like Sieve, Excalibur did not progress beyond a lab setting and was never actually deployed in a large-scale real-world setting.

Finally, most existing access-control applications separate the cryptographic and access-control aspects. Cryptography is used to protect sensitive data (in transit, but sometimes, also at rest), while access control is enforced more traditionally. In traditional models, a central authority typically enforces policies by validating an entity's authorization prior to granting data access. When access is granted, a decryption key is shared with the requesting entity, who can then access the data by decrypting it. However, this approach introduces significant latency to the access request. Furthermore, a single central authority makes the system vulnerable to denial-of-service attacks. In contrast, using ABE allows access control on a cryptographic level, achieving both protection of the data whilst allowing the enforcement of access control. By extension, it removes this extra latency and mitigates the availability issues of the central authority. It also allows data to be freely transmitted between nodes, removing the risk of accidental data leakage as long as the target node's secret key is not compromised.

10 Conclusion

For several months, Portunus has seen real-world usage protecting customer keys. It has succeeded in supplanting the capabilities of the legacy system and setting a foundation for future product development. This required multiple person-years of effort by a small team, as well as accepting a fairly novel scheme as the foundation of its security. This effort brought about increases in reliability and enhanced performance.

Acknowledgements

We are grateful for the assistance of the Security Engineering team, particularly Mihir Jham and Nicky Semenza, in explaining their software and letting us make some rather audacious proposals to redesign a significant system. We would also like to thank the Research team. Finally, Mike Rosulek helped us solve the hardest problem in Computer Science by suggesting the name Portunus[6].

[6]The Roman God of keys, thresholds, gates and ports.

References

[1] Cloudflare interoperable, reusable cryptographic library. https://pkg.go.dev/github.com/cloudflare/circl@v1.3.0/abe/cpabe/tkn20. Accessed: 2022-12-14.

[2] Salt project documentation. https://docs.saltproject.io/en/latest/. Accessed: 2022-10-25.

[3] Joseph A. Akinyele, Matthew W. Pagano, Matthew D. Green, Christoph U. Lehmann, Zachary N. J. Peterson, and Aviel D. Rubin. Securing electronic medical records using attribute-based encryption on mobile devices. In Xuxian Jiang, Amiya Bhattacharya, Partha Dasgupta, and William Enck, editors, *SPSM'11*, pages 75–86. ACM, 2011.

[4] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[5] Nuttapong Attrapadung and Junichi Tomida. Unbounded dynamic predicate compositions in ABE from standard assumptions. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT*, volume 12493 of *LNCS*, pages 405–436. Springer, 2020.

[6] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN*, volume 2576 of *LNCS*, pages 257–267. Springer, 2002.

[7] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS*, pages 62–73. ACM, 1993.

[8] Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT benchmarking of cryptographic systems. https://bench.cr.yp.to/results-encrypt.html. Accessed: 2022-10-25.

[9] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *S&P*, pages 321–334. IEEE, 2007.

[10] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *LNCS*, pages 223–238. Springer, 2004.

[11] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[12] Dan Boneh and Jonathan Katz. Improved efficiency for cca-secure cryptosystems built using identity-based encryption. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, pages 87–103, 2005. https://www.cs.umd.edu/~jkatz/papers/id-cca-mac.pdf.

[13] S. Bowe. BLS12-381: New zk-SNARK elliptic curve construction. https://blog.z.cash/new-snark-curve/.

[14] Sean Bowe. Faster subgroup checks for bls12-381. Cryptology ePrint Archive, Paper 2019/814, 2019. https://eprint.iacr.org/2019/814.

[15] Melissa Chase. Multi-authority attribute based encryption. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 515–534. Springer, 2007.

[16] Jie Chen, Romain Gay, and Hoeteck Wee. Improved dual system ABE in prime-order groups via predicate encodings. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT*, volume 9057 of *LNCS*, pages 595–624. Springer, 2015.

[17] Lily Chen. Recommendation for key derivation using pseudorandom functions. Technical Report NIST Special Publication (SP) 800-108, Rev. 1, National Institute of Standards and Technology, Gaithersburg, MD, 2022.

[18] Antonio de la Piedra, Marloes Venema, and Greg Alpár. ABE squared: Accurately benchmarking efficiency of attribute-based encryption. *TCHES*, 2022(2):192–239, 2022.

[19] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[20] Andres Erbsen, Jade Philipoom, Jason Gross, Robert Sloan, and Adam Chlipala. Simple high-level code for cryptographic arithmetic - with proofs, without compromises. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1202–1219. IEEE, 2019.

[21] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge L. Villar. An algebraic framework for diffie-hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO*, volume 8043 of *Lecture Notes in Computer Science*, pages 129–147. Springer, 2013.

[22] ETSI. ETSI TS 103 458 (V1.1.1). Technical specification, European Telecommunications Standards Institute (ETSI), 2018.

[23] ETSI. ETSI TS 103 532 (V1.1.1). Technical specification, European Telecommunications Standards Institute (ETSI), 2018.

[24] Armando Faz-Hernandez, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. Hashing to Elliptic Curves. RFC 9380, June 2023. https://doi.org/10.17487/rfc9380.

[25] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer, 1993.

[26] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO*, volume 196 of *LNCS*, pages 10–18. Springer, 1984.

[27] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *CCS*, pages 89–98. ACM, 2006.

[28] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. Cryptology ePrint Archive, Paper 2006/309, 2006. https://eprint.iacr.org/2006/309.

[29] R Granger and N. P. Smart. On computing products of pairings. Cryptology ePrint Archive, Paper 2006/172, 2006. https://eprint.iacr.org/2006/172.

[30] Javier Herranz. Attacking pairing-free attribute-based encryption schemes. *IEEE Access*, 8:222226–222232, 2020.

[31] Susan Hohenberger and Brent Waters. Online/offline attribute-based encryption. In Hugo Krawczyk, editor, *PKC*, volume 8383 of *LNCS*, pages 293–310. Springer, 2014.

[32] Seny Kamara and Kristin E. Lauter. Cryptographic cloud storage. In Radu Sion, Reza Curtmola, Sven Dietrich, Aggelos Kiayias, Josep M. Miret, Kazue Sako, and Francesc Sebé, editors, *FC*, volume 6054 of *LNCS*, pages 136–149. Springer, 2010.

[33] Eike Kiltz and Yevgeniy Vahlis. CCA2 secure IBE: standard model efficiency through authenticated symmetric encryption. In Tal Malkin, editor, *CT-RSA*, volume 4964 of *LNCS*, pages 221–238. Springer, 2008.

[34] Eike Kiltz and Yevgeniy Vahlis. Cca2 secure ibe: Standard model efficiency through authenticated symmetric encryption. Cryptology ePrint Archive, Paper 2008/020, 2008. https://eprint.iacr.org/2008/020.

[35] Lucas Kowalczyk and Hoeteck Wee. Compact adaptively secure ABE for \mathsf nc^1 from k-lin. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT*, volume 11476 of *LNCS*, pages 3–33. Springer, 2019.

[36] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. Cryptology ePrint Archive, Paper 2010/351, 2010. https://eprint.iacr.org/2010/351.

[37] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588. Springer, 2011.

[38] Vincent Hu (NIST), David Ferraiolo (NIST), Richard Kuhn (NIST), Adam Schnitzer (BAH), Kenneth Sandlin (MITRE), Robert Miller (MITRE), and Karen Scarfone (Scarfone Cybersecurity). Guide to attribute based access control (abac) definition and considerations. Technical report, National Institute of Standards and Technology, 2014. https://doi.org/10.6028/NIST.SP.800-162.

[39] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *LNCS*, pages 191–208. Springer, 2010.

[40] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *CCS*, pages 195–203. ACM, 2007.

[41] Ruoming Pang, Ramon Caceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner, Jeffrey L. Korn, Abhishek Parmar, Christina D. Richards, and Mengzhi Wang. Zanzibar: Google's consistent, global authorization system. In *2019 USENIX Annual Technical Conference (USENIX ATC '19)*, Renton, WA, 2019. https://research.google/pubs/pub48190/.

[42] Matthew Pirretti, Patrick Traynor, Patrick D. McDaniel, and Brent Waters. Secure attribute-based systems. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *CCS*, pages 99–112. ACM, 2006.

[43] Geoffrey Plouviez. Introducing quicksilver: Configuration distribution at internet scale.

https://blog.cloudflare.com/introducing-quicksilver-configuration-distribution-at-internet-scale, 2020. Accessed: 2022-10-25.

[44] Huiling Qian, Jiguo Li, Yichen Zhang, and Jinguang Han. Privacy-preserving personal health record using multi-authority attribute-based encryption with revocation. *Int. J. Inf. Secur.*, 14(6):487–497, nov 2015.

[45] Markku-Juhani O. Saarinen and Jean-Philippe Aumasson. The BLAKE2 cryptographic hash and message authentication code (MAC). Technical Report 7693, 2015. https://www.rfc-editor.org/rfc/rfc7693.

[46] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 457–473. Springer, 2005.

[47] Yumi Sakemi, Tetsutaro Kobayashi, Tsunekazu Saito, and Riad S. Wahby. Pairing-Friendly Curves. Internet-Draft draft-irtf-cfrg-pairing-friendly-curves-11, Internet Engineering Task Force, November 2022. Work in Progress.

[48] Nuno Santos, Rodrigo Rodrigues, Krishna P. Gummadi, and Stefan Saroiu. Policy-sealed data: A new abstraction for building trusted cloud services. In Tadayoshi Kohno, editor, *USENIX Security Symposium*, pages 175–188. USENIX Association, 2012.

[49] Michael Scott. MIRACL cryptographic SDK: Multiprecision Integer and Rational Arithmetic Cryptographic Library. https://github.com/miracl/MIRACL, 2003.

[50] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[51] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984.

[52] Nick Sullivan. Geo key manager: How it works. https://blog.cloudflare.com/geo-key-manager-how-it-works/, 2017. Accessed: 2022-10-25.

[53] Nick Sullivan and Brendan McMillion. Geo key manager. *Real World Crypto 2018*, jan 2018. https://rwc.iacr.org/2018/Slides/Sullivan.pdf.

[54] Junichi Tomida, Yuto Kawahara, and Ryo Nishimaki. Fast, compact, and expressive attribute-based encryption. Cryptology ePrint Archive, Paper 2019/966, 2019. https://eprint.iacr.org/2019/966.

[55] Junichi Tomida, Yuto Kawahara, and Ryo Nishimaki. Fast, compact, and expressive attribute-based encryption. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC*, volume 12110 of *LNCS*, pages 3–33. Springer, 2020.

[56] Junichi Tomida, Yuto Kawahara, and Ryo Nishimaki. Fast, compact, and expressive attribute-based encryption. *Des. Codes Cryptogr.*, 89(11):2577–2626, 2021.

[57] Marloes Venema and Greg Alpár. A bunch of broken schemes: A simple yet powerful linear approach to analyzing security of attribute-based encryption. In Kenneth G. Paterson, editor, *CT-RSA*, volume 12704 of *LNCS*, pages 100–125. Springer, 2021.

[58] Marloes Venema and Greg Alpár. GLUE: generalizing unbounded attribute-based encryption for flexible efficiency trade-offs. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 652–682. Springer, 2023.

[59] Marloes Venema, Greg Alpár, and Jaap-Henk Hoepman. Systematizing core properties of pairing-based attribute-based encryption to uncover remaining challenges in enforcing access control in practice. *Des. Codes Cryptogr.*, 91(1):165–220, 2023.

[60] Marloes Venema and Leon Botros. Efficient and generic transformations for chosen-ciphertext secure predicate encryption. Cryptology ePrint Archive, Paper 2022/1436, 2022.

[61] Frank Wang, James Mickens, Nickolai Zeldovich, and Vinod Vaikuntanathan. Sieve: Cryptographically enforced access control for user data in untrusted clouds. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 611–626, Santa Clara, CA, March 2016. USENIX Association.

[62] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *LNCS*, pages 619–636. Springer, 2009.

[63] David Wragg. Unimog —- cloudflare's edge load balancer. https://blog.cloudflare.com/unimog-cloudflares-edge-load-balancer/, 2020. Accessed: 2022-12-15.

[64] Shota Yamada, Nuttapong Attrapadung, Goichiro Hanaoka, and Noboru Kunihiro. Generic constructions for chosen-ciphertext secure attribute based encryption.

In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC*, volume 6571 of *LNCS*, pages 71–89. Springer, 2011.

[65] Kan Yang, Xiaohua Jia, Kui Ren, Bo Zhang, and Ruitao Xie. DAC-MACS: effective data access control for multiauthority cloud storage systems. *IEEE Trans. Inf. Forensics Secur.*, 8(11):1790–1801, 2013.

[66] Xuanxia Yao, Zhi Chen, and Ye Tian. A lightweight attribute-based encryption scheme for the internet of things. *Future Gener. Comput. Syst.*, 49:104–112, 2015.

A Security Model for CP-ABE

We define the security game IND-CCA(λ) between challenger and attacker as follows:

- **Setup phase:** The challenger runs Setup(λ) to obtain MPK and MSK, and sends the master public key MPK to the attacker.
- **First query phase:** The attacker can make two types of queries:
  - **Key query:** The attacker queries secret keys for sets of attributes S, and obtains $\mathrm{SK}_S \leftarrow \mathrm{KeyGen}(\mathrm{MSK}, S)$ in response.
  - **Decryption query:** The attacker sends a ciphertext $\mathrm{CT}_{\mathbb{A}}$ for access policy $\mathbb{A}$ and some set S that satisfies $\mathbb{A}$ to the challenger, who returns the message $M \leftarrow \mathrm{Decrypt}(\mathrm{MPK}, \mathrm{SK}_S, \mathrm{CT}_{\mathbb{A}})$ (where $\mathrm{SK}_S \leftarrow \mathrm{KeyGen}(\mathrm{MSK}, S)$).
- **Challenge phase:** The attacker specifies some access policy $\mathbb{A}^*$ such that none of the sets S in the first key query phase satisfies $\mathbb{A}^*$, generates two equal-length messages M_0 and M_1, and sends these to the challenger. The challenger flips a coin, i.e., $\beta \in_R \{0,1\}$, encrypts M_β under $\mathbb{A}^*$, i.e., $\mathrm{CT}_{\mathbb{A}^*} \leftarrow \mathrm{Encrypt}(\mathrm{MPK}, \mathbb{A}^*, M_\beta)$, and sends the resulting ciphertext $\mathrm{CT}^*_{\mathbb{A}^*}$ to the attacker.
- **Second query phase:** This phase is identical to the first query phase, with the additional restriction that the attacker cannot query keys for sets of attributes S that satisfy the policy $\mathbb{A}^*$ or make a decryption query for $\mathrm{CT}^*_{\mathbb{A}^*}$.
- **Decision phase:** The attacker outputs a guess β' for β.

The advantage of the attacker is defined as

$$\mathsf{Adv}_{\text{IND-CCA}} = \left| \Pr[\beta' = \beta] - \frac{1}{2} \right|.$$

A scheme is fully secure against chosen-ciphertext attacks if all polynomial-time attackers have at most a negligible advantage in this security game.

In the model for security against chosen-plaintext attacks, the attacker is not allowed to make decryption queries in the first and second query phase—only key queries.

B Other Definitions

B.1 Symmetric Encryption

B.1.1 Formal Definition

We define symmetric encryption as follows. Let λ be the security parameter. A symmetric encryption scheme $\mathrm{SE} = (\mathrm{Enc}, \mathrm{Dec})$, with symmetric key $\mathrm{K} \in \{0,1\}^\lambda$, is defined as

- $\mathrm{Enc}_\mathrm{K}(M)$: On input message $M \in \{0,1\}^*$, encryption returns a ciphertext $\mathrm{CT}_{\mathrm{sym}}$.
- $\mathrm{Dec}_\mathrm{K}(\mathrm{CT}_{\mathrm{sym}})$: On input ciphertext $\mathrm{CT}_{\mathrm{sym}}$, decryption returns a message M or an error message $\perp$.

The scheme is correct if for all keys $\mathrm{K} \in \{0,1\}^\lambda$ and all messages $M \in \{0,1\}^*$, we have $\mathrm{Dec}_\mathrm{K}(\mathrm{Enc}_\mathrm{K}(M)) = M$.

B.1.2 Security Model

For symmetric encryption, we use the same security notion as in [33], i.e., ciphertext indistinguishability. Informally, ciphertext indistinguishability ensures that an attacker cannot distinguish between encryptions of any two messages. More formally, it is defined as follows. Let λ be a security parameter and let $\mathrm{SE} = (\mathrm{Enc}, \mathrm{Dec})$ be a symmetric encryption scheme. Consider the following game between a challenger and attacker. The challenger first picks a key $\mathrm{K} \in \{0,1\}^\lambda$. Then, the attacker specifies two messages M_0, M_1 and gives these to the challenger, who flips a coin $\beta \in_R \{0,1\}$ and returns $\mathrm{CT}_{\mathrm{sym}} \leftarrow \mathrm{Enc}_\mathrm{K}(M_\beta)$ to the attacker. The attacker outputs a guess β' for β. Then, $\mathrm{SE} = (\mathrm{Enc}, \mathrm{Dec})$ has indistinguishable ciphertexts if for all polynomial-time attackers in the game above holds that the advantage $\left| \Pr[\beta' = \beta] - \frac{1}{2} \right|$ is negligible

B.2 MAC Function

B.2.1 Formal Definition

We formally define a MAC function as follows. Let λ be the security parameter. A message authentication code (MAC) $(\mathrm{MAC}_{\mathrm{K_{MAC}}}, \mathrm{Vrfy}_{\mathrm{K_{MAC}}})$, where $\mathrm{K_{MAC}} \in \{0,1\}^\lambda$ is the MAC key, is defined by

- $\mathrm{MAC}_{\mathrm{K_{MAC}}}(M)$: On input message $M \in \{0,1\}^*$, this algorithm outputs a tag T.
- $\mathrm{Vrfy}_{\mathrm{K_{MAC}}}(M, \mathrm{T})$: On input message M and tag T, the algorithm returns 0 (“reject”) or 1 (“accept”).

The MAC is correct if for all keys $\mathrm{K_{MAC}} \in \{0,1\}^\lambda$ and all messages $M \in \{0,1\}^*$ it holds that if $\mathrm{T} \leftarrow \mathrm{MAC}_{\mathrm{K_{MAC}}}(M)$, then $\mathrm{Vrfy}_{\mathrm{K_{MAC}}}(M, \mathrm{T}) = 1$.

B.2.2 Security Model

For MACs, we use the notion of security against one-time chosen-message attacks. Let λ be the security parameter, and let $(\mathrm{MAC}_{\mathrm{K_{MAC}}}, \mathrm{Vrfy}_{\mathrm{K_{MAC}}})$ be a message authentication code. Consider the following game between challenger and attacker. The challenger first picks a key $\mathrm{K_{MAC}} \in \{0,1\}^{\lambda}$. The attacker sends a message M to the challenger, who returns a tag $\mathrm{T} \leftarrow \mathrm{MAC}_{\mathrm{K_{MAC}}}(M)$. Then, the attacker outputs a pair (M', T'). The attacker succeeds if $(M, \mathrm{T}) \neq (M', \mathrm{T}')$ and $\mathrm{Vrfy}(M', \mathrm{T}') = 1$.

B.3 Special Commitment Scheme

B.3.1 Formal Definition

The special commitment scheme that we use is defined as follows. Let λ be the security parameter.

- $\mathrm{ESetup}(\lambda) \rightarrow \mathrm{pub}$: Define hashes $h_1 \colon \{0,1\}^{448} \rightarrow \mathbb{Z}_p$ and $h_2 \colon \{0,1\}^{448} \rightarrow \{0,1\}^{\lambda}$, and set $\mathrm{pub} = (h_1, h_2)$.

- $\mathrm{ES}(\lambda, \mathrm{pub}) \rightarrow (\mathrm{rand}, \mathrm{com}, \mathrm{dec})$: Generate $\mathrm{dec} \in_R \{0,1\}^{448}$, and compute $\mathrm{com} = h_1(\mathrm{dec})$ and $\mathrm{rand} = h_2(\mathrm{dec})$.

- $\mathrm{ER}(\mathrm{pub}, \mathrm{com}, \mathrm{dec}) \rightarrow \mathrm{rand}$: Generate $\mathrm{rand} \leftarrow h_2(\mathrm{dec})$.

The special commitment scheme is correct if for all $(\mathrm{rand}, \mathrm{com}, \mathrm{dec}) \leftarrow \mathrm{ES}(\lambda, \mathrm{pub})$ holds that $\mathrm{ER}(\mathrm{pub}, \mathrm{com}, \mathrm{dec}) = \mathrm{rand}$.

B.3.2 Security Model

A special commitment scheme $(\mathrm{ESetup}, \mathrm{ES}, \mathrm{ER})$ is secure if it is hiding and binding.

- **Hiding:** Consider an attacker and a challenger. Then, the challenger runs $\mathrm{pub} \leftarrow \mathrm{ESetup}(\lambda)$ and flips a coin $\beta \in_R \{0,1\}$. If $\beta = 0$, then $\mathcal{C}$ generates a random $\mathrm{rand} \in_R \{0,1\}^{\lambda}$, and otherwise, it runs $(\mathrm{rand}, \mathrm{com}, \mathrm{dec}) \leftarrow \mathrm{ES}(\lambda, \mathrm{pub})$. It shares $(\lambda, \mathrm{pub}, \mathrm{rand}, \mathrm{com})$ with the attacker, who then outputs a guess β' for β. The scheme is hiding if for all such attackers, it holds that the advantage $\left|\Pr[\beta = \beta'] - \frac{1}{2}\right|$ is negligible.

- **Binding:** Consider an attacker and a challenger. Then, the challenger runs $\mathrm{pub} \leftarrow \mathrm{ESetup}(\lambda)$, and shares $(\mathrm{rand}, \mathrm{com}, \mathrm{dec}) \leftarrow \mathrm{ES}(\lambda, \mathrm{pub})$ with the attacker. Then, it is computationally infeasible for the attacker to find $\mathrm{dec}' \neq \mathrm{dec}$ such that $\mathrm{ER}(\mathrm{pub}, \mathrm{com}, \mathrm{dec}') = \mathrm{rand}$, i.e., for output $\mathrm{dec}' \neq \mathrm{dec}$ of the attacker, it holds that the success probability $\Pr[\mathrm{ER}(\mathrm{pub}, \mathrm{com}, \mathrm{dec}') = \mathrm{rand}]$ is negligible. The scheme is binding if this holds for all such attackers.

C Description of Our CCA-Secure Scheme

We give a simplified description (using the simplified description of TKN20 in Section 3.2.10) of our CCA-secure scheme below.

- $\mathrm{Setup}(\lambda) \rightarrow (\mathrm{MPK}, \mathrm{MSK})$: The setup outputs the master public-secret key pair (MPK, MSK), where $H_i \colon \{0,1\}^* \rightarrow \mathbb{G}_1$ with $i \in \{0,1\}$ are two hash functions (modeled as random oracles), $\mathrm{KDF} : \mathbb{G}_T \rightarrow \{0,1\}^{\lambda}$ is a key derivation function [17], $\mathrm{SE} = (\mathrm{Enc}, \mathrm{Dec})$ is a symmetric encryption scheme, pub are the public parameters generated with the setup ESetup of a special commitment scheme, $\mathrm{MSK} = (\alpha, b)$, and

$$\mathrm{MPK} = (\mathrm{SE}, \mathrm{pub}, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, H_0, H_1, A = e(g_1, g_2)^{\alpha}, B = g_1^b).$$

- $\mathrm{KeyGen}(\mathrm{MSK}, (S, \psi_{\mathrm{lab}})) \rightarrow \mathrm{SK}_S$: On input a set of attribute values S and the associated labeling map $\psi_{\mathrm{lab}} \colon S \rightarrow \{0,1\}^*$, which maps the attributes in the set S to labels (represented as strings), it outputs the secret key SK_S as

$$\mathrm{SK}_S = (S, K_1 = g_1^{\alpha + rb}, K_2 = g_2^r,$$
$$\{K_{3,\mathrm{att}} = (H_0(\psi_{\mathrm{lab}}(\mathrm{att})) \cdot H_1(\psi_{\mathrm{lab}}(\mathrm{att}))^{x_{\mathrm{att}}})^r\}_{\mathrm{att} \in S},$$
$$K_{3,\mathrm{CCA}} = H_0(\mathrm{CCA})^r, K_{4,\mathrm{CCA}} = H_1(\mathrm{CCA})^r),$$

  where $r \in_R \mathbb{Z}_p$ is a randomly generated element in $\mathbb{Z}_p$ and x_{att} denotes the representation of att in $\mathbb{Z}_p$.

- $\mathrm{Encrypt}(\mathrm{MPK}, \mathbb{A}, M) \rightarrow \mathrm{CT}_{\mathbb{A}}$: On input a plaintext message $M \in \{0,1\}^*$ and an access policy $\mathbb{A} = (\mathbf{A}, \rho, \rho_{\mathrm{lab}}, \overline{\rho}, \tau)$—where $\tau \colon \{1, \ldots, n_1\} \rightarrow \{1, \ldots, m\}$ is a function that maps each row that is associated with the same label to a different integer in $\{1, \ldots, m\}$, with m being the maximum number of times that a label occurs in the policy—it first extends the policy $\mathbb{A}$ to $\mathbb{A}'$ such that it applies an AND operator to $\mathbb{A}$ and the attribute label-value pair `CCA: com` (where com is defined as below), and outputs a ciphertext $\mathrm{CT}'_{\mathbb{A}'}$ as

$$\mathrm{CT}'_{\mathbb{A}'} = (\mathbb{A}, \mathrm{com} \leftarrow h_2(\mathrm{dec}), C \leftarrow \mathrm{Enc}_{\mathrm{K}}(\mathrm{dec} \| M), \mathrm{CT}_{\mathbb{A}'},$$
$$\mathrm{T} = \mathrm{MAC}_{\mathrm{K}'}(\mathbb{A} \| \mathrm{com} \| C \| \mathrm{CT}_{\mathbb{A}'})),$$

  so that

$$\mathrm{CT}_{\mathbb{A}} = (C_1 = g_2^s, \{C_{2,l} = g_2^{s_l}\}_{l \in \{1, \ldots, m\}},$$
$$\left\{C_{3,j} = B^{\lambda_j} \cdot (H_0(\rho_{\mathrm{lab}}(j)) \cdot H_1(\rho_{\mathrm{lab}}(j))^{x_{\rho(j)}})^{s_{\tau(j)}}\right\}_{j \in \chi_0},$$
$$\left\{C_{3,j} = B^{-\lambda_j} \cdot H_0(\rho_{\mathrm{lab}}(j))^{s_{\tau(j)}},\right.$$
$$\left. C_{4,j} = B^{x_{\rho(j)} \lambda_j} \cdot H_1(\rho_{\mathrm{lab}}(j))^{s_{\tau(j)}}\right\}_{j \in \chi_1}),$$

  where $s, s_1, \ldots, s_m, v_2, \ldots, v_{n_2+1} \in_R \mathbb{Z}_p$ are randomly generated elements in $\mathbb{Z}_p$, $\lambda_j = A_{j,1} s + \sum_{k \in \{2, \ldots, n_2+1\}} A_{j,k} v_k$, $\chi_i = \{j \in \{1, \ldots, n_1 + 1\} \mid \overline{\rho}(j) = i\}$ for $i \in \{0,1\}$, $\mathrm{K} \leftarrow \mathrm{KDF}(A^s)$, $\mathrm{dec} \in_R \{0,1\}^{448}$ and $\mathrm{K}' \leftarrow h_1(\mathrm{dec})$.

- Decrypt$(\mathrm{SK}_S, \mathrm{CT}_{\mathbb{A}'}) \rightarrow M'$: On input the ciphertext $\mathrm{CT}_{\mathbb{A}'}$ (where $\mathbb{A}'$ is an AND-composition of policy $\mathbb{A}$ and `CCA: com`), and a secret key SK_S, it first checks whether S satisfies the $\mathbb{A}$. If not, then it aborts. Otherwise, it first determines $\Upsilon_0 = \{j \in \chi_0 \mid \rho(j) \in S\}$, $\Upsilon_1 = \{j \in \chi_1 \mid \rho(j) \notin S \wedge \rho_{\mathrm{lab}}(j) \in \psi_{\mathrm{lab}}(S)\}$, $\Upsilon = \Upsilon_0 \cup \Upsilon_1$ and $\{\varepsilon_j\}_{j\in\Upsilon}$ such that $\sum_{j\in\Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$, then computes $e(g_1, g_2)^{\alpha s}$ as in the decryption of TKN20 (Section 3.2.10), where a key can be generated for `CCA : com` by computing $K_{3,\mathrm{CCA}} \cdot K_{4,\mathrm{CCA}}^{x_{\mathrm{com}}}$, then retrieves:

$$\begin{aligned} \mathrm{K} &\leftarrow \mathrm{KDF}(e(g_1,g_2)^{\alpha s}) \\ \mathrm{dec} \| M &\leftarrow \mathrm{Dec}_{\mathrm{K}}(C) \\ \mathrm{K}' &\leftarrow h_1(\mathrm{dec}), \end{aligned}$$

and verifies:

$$\begin{aligned} h_2(\mathrm{dec}) &\stackrel{?}{=} \mathrm{com} \\ \mathrm{Vrfy}(\mathbb{A} \| \mathrm{com} \| C \| \mathrm{CT}_{\mathbb{A}'}, \mathrm{T}) &\stackrel{?}{=} 1. \end{aligned}$$

If both checks pass, then the decryption returns M, and if not, it returns an error message.