Portunus: Re-imagining Access Control in Distributed Systems

Watson Ladd
Akamai Technologies *

Marloes Venema
University of Wuppertal and Radboud University

Tanya Verma
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 customizing 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. We have deployed Portunus on Cloudflare's global network of over 400 data centers. Our measurements indicate that we can handle millions of requests per second globally, making it one of the largest deployments of ABE.

1 Introduction

Globally distributed infrastructure providers, particularly content delivery networks (CDNs), perform TLS termination on behalf of their clients in order to improve performance and protect client web properties against attacks. To handle this TLS termination, providers must have a method of securing their clients' private signing keys, as well as distributing keys to the edge servers that handle user traffic in order to keep handshake latency low.

Infrastructure providers that operate at scale, however, operate a network of data centers that span a wide range of countries, hardware, localized legal and compliance domains, and degrees of control over data center environment. What one customer needs from a server to trust it with their key does not match what another needs. Centralized methods of key management or access control [37] force an extra round-trip to complete requests, creating high latency overheads and reducing reliability. Using standard public-key encryption produces a key management problem as well as forces every customer key to be encrypted many times, expanding the volume of data to handle. It also introduces operational issues as keys must be re-encrypted as servers are added.

Because storage space in a global key store is expensive, we looked for a more direct way to solve these access-control issues through cryptography, without imposing the overheads of a centralized service or the space expansion and key management and operational issues of a naive cryptographic approach.

Cloudflare's first attempt at a solution leveraged a combination of identity-based encryption [11,46] and broadcast encryption [23] to storing customer private keys [47]. Unfortunately, the tailor-made cryptographic protection mechanism we built created barriers: it had an inflexible set of applicable access restrictions, consisting of an allowed or excluded list of regions. As new data centers were added, they were not able to decrypt old keys because of this inflexibility and, thus, did not improve the performance of the system as much as expected. In addition, we learned that changing the assignment of attributes incurred a high management overhead, which limited our ability to dynamically react to varying customer and compliance needs.

Spurred by these restrictions, we overhauled much of the internals of this solution and created Portunus. Underlying Portunus is ciphertext-policy attribute-based encryption (CP-ABE) [9], which can implement fine-grained access

control on a cryptographic level. ABE was first proposed by Sahai and Waters [42] 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) [34].

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 Quicksilver. 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 central certificate authority, 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. Another ancillary benefit to using ABE is that each machine's attribute secret key is unique, reducing the risk of immediate global compromise that arises from sharing the same decryption key between every machine.

Adopting CP-ABE as a storage layer 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 [21, 22, 29, 43], 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 future ABE researchers about how ABE works in the real world

2 Requirements

In the design process for Portunus, we informally identified a series of requirements arising from customer needs as well as internal engineering demands and the experience of operating the historic system, Geo Key Manager.

- **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 process of responding. Historically, such floods have caused significant incidents where data centers were unable to process handshakes fast enough due to unscalable networking logic.

- **Rotation capable:** It should be easy to rotate keys used in the system. A rotation ensures that newly-uploaded certificate 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. 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 has changed from what was envisioned, such as when a new compliance standard is introduced. Accommodating these changes in the prior system, Geo Key Manager, took considerable work. Furthermore, Geo Key Manager could not quickly respond to needs to remove TLS termination from certain machines or data centers.

- **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, the future developments would be difficult to anticipate.

- **Limited storage:** Quicksilver, Cloudflare's configuration management system [39], has limited space due to its fast replication strategy that 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 be cognizant of its storage overhead.

- **Uniformity of data:** Quicksilver depends on a homogeneous tree method of replication: 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. This means that each edge machine has the same view of the entire Quicksilver dataset.

3 Cryptographic Building Blocks

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` [36]. 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 it into the structures that are consumed by the ABE scheme 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 [42]. Unlike traditional public-key encryption, ABE allows users to enforce a more fine-grained access control to the encrypted data [2, 9, 25, 38, 52]. There are two variants of ABE: key-policy ABE (KP-ABE) [25], 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. We give a formal definition of CP-ABE below.

3.2.3 Formal Definition of CP-ABE

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

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

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)`	Don't decrypt in Russia and US
`country: US and security: high`	Decrypt only in US data centers with a high level of security

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 [52]. 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 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 [24]. 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 [44], RELIC [3] or our own library, CIRCL [17]. 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) [48–50]. We have open-sourced this code as part of our cryptographic library, CIRCL [17].

The reason why we chose the TKN20 scheme is because it is currently the only ABE scheme that has a full description and satisfies the following properties simultaneously [52]:

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 [26]. In such matrices, the rows of the matrix are associated with the attributes used in the policy. To

[1] In ABE, NOT operators can be supported in three ways [4]. TKN20 (and thus, our scheme) supports the most efficient variant proposed by Okamoto and Takashima [35]. 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]

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 is the one proposed by Lewko and Waters [33]. For example, the policy `role: doctor and region: EU` is represented as the pair $(\mathbf{A}, \rho)$ 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 [26] 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. [36], 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 be found in the TKN20 paper [49, 50]—can be constructed from the simplified version.

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

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

- $\text{KeyGen}(\text{MSK}, (S, \psi_{\text{lab}})) \to \text{SK}_S$: On input a set of attribute values S and the associated labeling map $\psi_{\text{lab}}: S \to \{0,1\}^*$, 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 $\mathbb{Z}_p$.

[3] This approach is also used in Shamir's secret sharing scheme [45].

- Encrypt$(\mathrm{MPK},\mathbb{A},M)\to \mathrm{CT}_{\mathbb{A}}$: On input a plaintext message $M\in\mathbb{G}_T$ and an access policy $\mathbb{A}=(\mathbf{A},\rho,\rho_{\mathrm{lab}},\overline{\rho},\tau)$—where $\tau\colon\{1,\ldots,n_1\}\to\{1,\ldots,m\}$ 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 $\mathrm{CT}_{\mathbb{A}}$ as

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

where $s,s_1,\ldots,s_m,v_2,\ldots,v_{n_2}\in_R\mathbb{Z}_p$ 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 $\chi_i=\{j\in\{1,\ldots,n_1\}\,|\,\overline{\rho}(j)=i\}$ for $i\in\{0,1\}$.

- Decrypt$(\mathrm{SK}_S,\mathrm{CT}_{\mathbb{A}})\to M'$: On input the ciphertext $\mathrm{CT}_{\mathbb{A}}$, and a secret key SK_S, it first checks whether S satisfies the $\mathbb{A}$. If not, then it aborts. Otherwise, it computes the message by first determining $\Upsilon_0=\{j\in\chi_0\mid\rho(j)\in S\}$, $\Upsilon_1=\{j\in\chi_1\mid\rho(j)\notin S\wedge\rho_{\mathrm{lab}}(j)\in\psi_{\mathrm{lab}}(S)\}$, $\Upsilon=\Upsilon_0\cup\Upsilon_1$ and $\{\varepsilon_j\}_{j\in\Upsilon}$ such that $\sum_{j\in\Upsilon}\varepsilon_j\mathbf{A}_j=(1,0,\ldots,0)$, 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 $y_j=x_{\psi_{\mathrm{lab}}^{-1}(\rho_{\mathrm{lab}}(j))}$. Then, $M=C/e(g_1,g_2)^{\alpha s}$. Note that this can be computed more efficiently by using the bilinearity property.

3.2.11 Description of the Fully Secure Variant

The structure of the actual TKN20 scheme [49] 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 [20]. This assumption is closely related to the decisional Diffie-Hellman assumption [19, 20]. The main technique that is used to achieve this level of security is the dual-system encryption technique [55]. Currently, the most advanced and efficient techniques [15, 32] in this paradigm use matrix structures "in the exponent", e.g., mapping the key component $K_1=g_1^{\alpha+rb}$ to $g_1^{\mathbf{a}+\mathbf{Wr}}$, where $\mathbf{a}$ and $\mathbf{r}$ are vectors of length 3 and $\mathbf{W}$ is a (3×3)-matrix [32].

3.2.12 Support for Wildcards

To support CCA-security more efficiently than e.g., [57], we use wildcards in the secret keys (as also proposed in the journal version of TKN20, i.e., [50]). 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:

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

Tomida et al. [50] 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 [16]. 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 [28, 30]. 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., [31, §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 [41], 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

[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.

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 [53] that this construction is CCA-secure.

3.3 API and Implementation

Listing 1: Example usage of the ABE library API

```
masterPublicKey, masterSecretKey := Setup()
policy := Policy{}
policy.FromString("country: US or region: EU")
ciphertext := masterPublicKey
    .Encrypt(policy, []byte("secret"))
attributesParisDC := Attributes{}
attributesParisDC.FromMap(map[
    string]string{"country": "FR", "region": "EU"}
attributeSecretKeyParisDC
     := masterSecretKey.KeyGen(attributesParisDC)
plaintext :=
     attributeSecretKeyParisDC.Decrypt(ciphertext)
assertEquals(plaintext, "secret")
```

We implemented our scheme as part of the CIRCL library [17] 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]. 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., $\mathbb{G}_1$. We have also optimized the arithmetic through judicious choice of representation.

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 Certificate Authority (CA).

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.

Table 4 in Appendix D provides a summary of the various keys.

4.2 Accessing Customer Keys

On receipt of a connection to a site, such as `alice.example.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 neighboring one, and forwards the request there. Gokeyless on this machine then decrypts the policy key and uses the result to

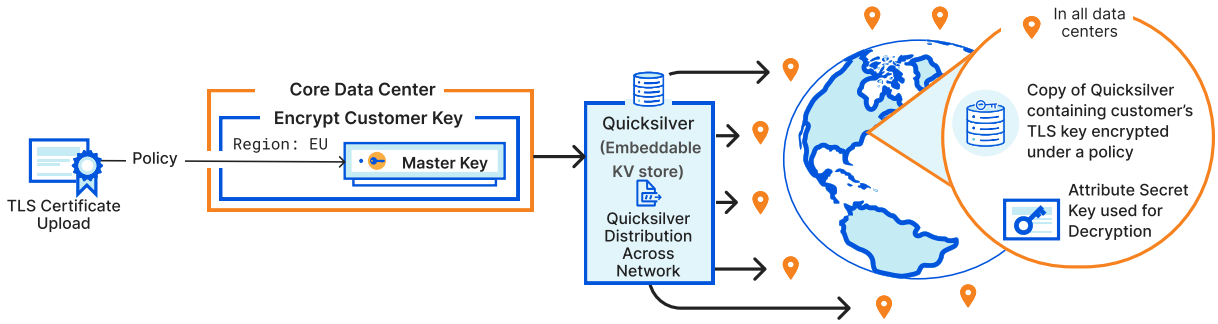

Figure 1: Encryption under a policy

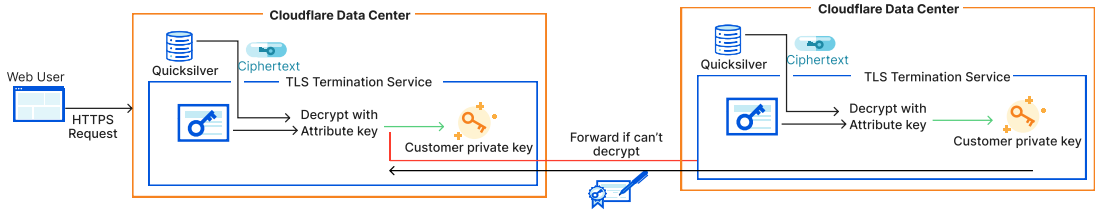

Figure 2: Decryption

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 certificate authority (CA) 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 [1]. 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 certificate 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 TLS keys post-rotation via access to Quicksilver.

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 need to be made to ensure that the system remains functional [4, 52]. To explain what needs to be changed, we split the act of "changing an attribute" in two steps: the removal of the old label and value and the re-addition of the label with the new value. First, removing the old attribute means that the associated data centers lose some decryption capabilities, because they do not satisfy the policies that required the presence of this label anymore. 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, because of the semantics of negation.

Carrying out this transition requires three steps. First, the affected label is removed from the forwarding information for the affected 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 the requests are handled by the data center again. The data center affected is able to handle 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

One of the motivations for the introduction of Portunus was a series of incidents in which the underlying RPC protocol permitting Gokeyless to forward requests to remote centers was proved insufficient, as well as highlighting the inflexibility of past approaches to key distribution.

Gokeyless primarily uses a custom binary protocol designed for simple clients in languages such as C. The protocol is a simple request-response protocol: requests have IDs permitting responses to come out of order, and packets are tagged length-value pairs. Because maintaining connections to all other machines is expensive and unnecessary, machines within one data center will elect among themselves a machine to forward requests to foreign data centers. This reduces the number of TCP connections being used.

To enable this, the built-in `net/rpc` package is used with a custom encoder built on top of the binary protocol to enable the Geo Key Manager functionality. Unfortunately, this library has an internal lock taken by a request and that is not released until the response is read. Until we had tracing the impact of this lock on performance was difficult to see, since many forwarded requests did not encounter this behavior. Once tracing was added, the lock was quickly visible and the integration with the net/rpc package changed to avoid it.

There is no sophisticated client-side load balancing: requests are forwarded to the closest satisfying data center, which on arrival leverage a network layer load balancer [56] 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.

Resiliency of the Cloudflare network 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, particularly in the case of low capacity data centers. If all data centers a customers key is decryptable in are offline, then the customer's website will be rendered inaccessible. To prevent this from happening, we require customer keys be decryptable in at least two large-capacity data centers.

4.7 Monitoring and Alerting

We have component-level metrics of decryptions, proxying and performance. We also have probes that measure test site uptimes by policy, and we have alerts when decryption failures hit a certain threshold. We have also integrated distributed tracing across Gokeyless: this has proven invaluable in finding sources of tail latency. Furthermore, 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.

5 Evaluation

While Portunus was launched to customers this year (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.

In a sample week in December 2022, Cloudflare was observed to handle over 8 million TLS requests per second globally across its fleet of data centers. Of these requests, over 20% used session resumption, which translates to approximately 6.5 million signatures per second being served. 100k rps are signatures using 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 Golang 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 a laptop with an Intel Core i7-10610U CPU @ 1.80GHz.

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 than necessary for any of Portunus' applications, as most policies are typically limited to a combination of geographic

Table 2: Space Overheads (bytes)

Scheme	Secret key[5]	Public key	Encrypt 23B	Encrypt 10KB
RSA-2048	1190	256	233	3568
X25519	32	32	48	48
Our scheme	33416	3282	19419	19419

Table 3: Operation times (ms)

Scheme	Key Gen.	Encrypt 23B	Decrypt 23B
RSA-2048	117	0.043	1.26
X25519	0.045	0.093	0.046
Our scheme	1796	704	62.4

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 4.1 rather than encrypting every customer key using ABE. It's also worth considering that the ciphertext in our system can be decrypted by a group of 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 core.

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 this 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 optimize the decryption process. We have discussed some of our optimizations for decryption in Section 3.3.

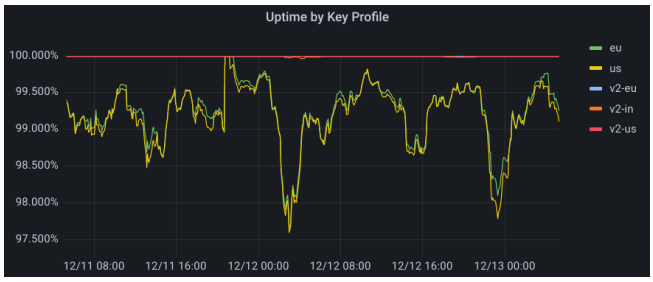

Figure 3: Uptime by policy, showing v2 having much better uptime than V1

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 that need to forwarded to a remote data center.

The vast majority of local unwrappings 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.

5.3 Analysis

Over the years that we have provided customers with these geographic limitation capabilities and from our implementation experience, we have learnt several lessons. As we discussed both central servers and directly encrypting secrets to the servers that need them have substantial downsides. Alternatively, modifying Quicksilver's replication strategy to store only a subset of the keyset on any given machine would challenge core design decisions Cloudflare has made over the years, which assume the entire dataset is replicated on every machine.

This encouraged us to explore cryptographic solutions. Our first attempt (Geo Key Manager) was back in 2016, when there were not many ABE schemes efficient and flexible enough to support our use case. In particular, ABE schemes supporting negations were rather impractical. Therefore, we initially used a combination of identity-based encryption and broadcast encryption to simulate an ABE-like scheme. However, this scheme was not collusion resistant (Section 3.2.4). Given that collusion-resistance is a definitional property of considering a scheme to be a valid ABE scheme, we were eager to switch

to a more theoretically satisfying solution once practical ABE schemes became available.

Once more practical ABE schemes became available, the difficulty in translating a scheme from an ABE paper to practice should not be underestimated, as well as selecting the appropriate scheme. 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.

Our scheme does not support policies with wildcards of the form `country: *`, which while not terribly limiting geographically are limitations in other contexts. It likewise does not permit an attribute set with multiple values for a single attribute label, such as `{group: fiddlers, group: percussionists}`.

While we performed a number of cryptographic optimizations 3.3, the implementation certainly has larger overheads compared to the mature implementations for traditional cryptographic schemes. We attempted to mitigate some of these costs by using policy keys 4.1. This is similar to hybrid encryption, where public key cryptography is used to establish a shared symmetric key, which then gets used to encrypt data. The difference here is that the policy keys are not symmetric, but rather X25519 key pairs, which is an asymmetric scheme based on elliptic curves. While not as fast as symmetric schemes like AES, traditional elliptic curve cryptography is significantly faster than attribute-based encryption. The advantage here is that the central service doesn't need access to secret key material to encrypt customer keys.

It is unclear what post-quantum ABE schemes will have the combination of performance, implementation simplicity, and expressivity required by this application. 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.

We have found that typically the certificates are stored in regions where many of the 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. It required a significant effort to add in tracing to diagnose and reproduce scaling issues due to long-held locks.

We are currently working on performing delegated credentials [5] in combination with the current system. This will enable us to reduce the additional latency incurred while forwarding requests to another data center. Delegated credentials allow short-lived credentials to be issued, so we can wrap the private keys and delegated credentials separately for separate data centers. We are also working on threshold signatures, where a customer's full private key will be inaccessible by every server. We can do this by encrypting the pieces of the private key using Portunus with attributes that correspond to certain security parameters.

Our adoption of an ABE based scheme instead of the earlier ad hoc construction lead directly to improvements to reliability as new data centers automatically contributed to the handling of remote signatures when able versus the earlier construction where the lists were largely static.

In short the adoption of Portunus has proved successful in increasing the reliability and enhancing the functionality of geographically oriented key management features.

6 Related Work

Portunus is an instantiation of distributed access control using a relatively under-utilized cryptographic construction.

CP-ABE papers will regularly start with a justification of their new capabilities based on use in system access control, e.g., [9, 40, 58] but without more details on the system design. Additionally, as Venema and Alpár demonstrate [51], there have been various attacks on some of their constructions [40, 58], particularly those that do not rely on pairings [27, 59]. Integrating a scheme that is broken into a real-world system is dangerous, as it allows adversaries to attack that system.

Sieve [54] dives deeper into the details, discussing how key management, handling ABE overhead, and deletion have to be addressed in a real system. Furthermore, it was integrated into real applications as a demonstration.

Excalibur [43] is perhaps the most similar to our work. They designed and implemented a system for customers to make data accessible on certain machines, albeit with a less expressive policy than ours, and integrated it with a cloud environment Eucalyptus. However, they did not progress beyond a lab setting and in their chosen application decryption latency and forwarding requests are not relevant.

7 Conclusion and Future Work

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, as well as uncovered and solved some longstanding bugs.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] Richard Barnes, Subodh Iyengar, Nick Sullivan, and Eric Rescorla. Delegated Credentials for (D)TLS. Internet-Draft draft-ietf-tls-subcerts-15, Internet Engineering Task Force, June 2022. Work in Progress.

[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] 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.

[16] 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.

[17] Cloudflare. *Cloudflare Interoperable, Reusable Cryptographic Library*. Accessed: 2022-12-14.

[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] 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.

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

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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/.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

[43] 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.

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

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

[46] 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.

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

[48] 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.

[49] 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.

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

[51] 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.

[52] 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.

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

[54] 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.

[55] 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.

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

[57] 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.

[58] 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.

[59] 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 CT_sym.
- $\mathrm{Dec}_\mathrm{K}(\mathrm{CT}_\mathrm{sym})$: On input ciphertext CT_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 [30], 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) $(\mathrm{MAC}_{\mathrm{K}_\mathrm{MAC}}, \mathrm{Vrfy}_{\mathrm{K}_\mathrm{MAC}})$, where $\mathrm{K}_\mathrm{MAC} \in \{0,1\}^\lambda$ is the MAC key, is defined by

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

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

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 $(\mathrm{MAC}_{\mathrm{K}_\mathrm{MAC}}, \mathrm{Vrfy}_{\mathrm{K}_\mathrm{MAC}})$ be a message authentication code. Consider the following game between challenger and attacker. The challenger first picks a key $\mathrm{K}_\mathrm{MAC} \in \{0,1\}^\lambda$. The attacker sends a message M to the challenger, who returns a tag $\mathrm{T} \leftarrow \mathrm{MAC}_{\mathrm{K}_\mathrm{MAC}}(M)$. 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 $h_1 : \{0,1\}^{448} \rightarrow \mathbb{Z}_p$ and $h_2 : \{0,1\}^{448} \rightarrow \{0,1\}^\lambda$, 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.

- 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), KDF : $\mathbb{G}_T \rightarrow \{0,1\}^\lambda$ is a key derivation function [16], SE $=$ (Enc, Dec) is a symmetric encryption scheme, pub are the public parameters generated with the setup ESetup of a special commitment scheme, MSK $= (\alpha, b)$, and

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

- KeyGen(MSK, (S, ψ_{lab})) $\rightarrow$ 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

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

  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 $\mathbb{Z}_p$.

- Encrypt(MPK, $\mathbb{A}$, M) $\rightarrow$ CT$_{\mathbb{A}}$: On input a plaintext message $M \in \{0,1\}^*$ and an access policy $\mathbb{A} = (\mathbf{A}, \rho, \rho_{\text{lab}}, \overline{\rho}, \tau)$—where $\tau\colon \{1,...,n_1\} \rightarrow \{1,...,m\}$ is a function that maps each row that is associated with the same label to a different integer in $\{1,...,m\}$, 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 CT$'_{\mathbb{A}'}$ as

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

  so that

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

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

- Decrypt(SK$_S$, CT$_{\mathbb{A}'}$) $\rightarrow M'$: On input the ciphertext 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 $\Upsilon_0 = \{j \in \chi_0 \mid \rho(j) \in S\}$, $\Upsilon_1 = \{j \in \chi_1 \mid \rho(j) \notin S \wedge \rho_{\text{lab}}(j) \in \psi_{\text{lab}}(S)\}$, $\Upsilon = \Upsilon_0 \cup \Upsilon_1$ and $\{\varepsilon_j\}_{j \in \Upsilon}$ such that $\sum_{j \in \Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$, 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 $K_{3,\text{CCA}} \cdot K_{4,\text{CCA}}^{x_{\text{com}}}$, then retrieves:

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

  and verifies:

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

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

D Table of Keys

Table 4 summarizes the keys that are used in Portunus.

Table 4: Summary of different keys

Key	Purpose	CA in core	Certificate Manager in core	Edge machine
Master Public Key	Encrypts private policy keys over an access policy	Generate	Read	
Master Secret Key	Generates secret keys for machines based on their attributes	Generate, Read		
Machine Secret Key / Attribute-Based Secret Key	Decrypts private policy keys stored in global key-value store, Quicksilver	Generate		Read
Customer TLS Private Key	Performs digital signature necessary to complete TLS handshake to the customer's website	Generate	Read (transiently on upload)	Read
Public Policy Key	Encrypts customers' TLS private keys		Generate, Read	
Private Policy Key	Decrypts customers' TLS private keys	Read (transiently during key rotation)	Generate	
Machine Identity Key	Authenticates machine to CA to securely fetch machine secret key	Read		Read