Cache-22: A Highly Deployable End-To-End Encrypted Cache System with Post-Quantum Security*

Keita Emura[§], Shiho Moriai[§], Takuma Nakajima[¶], and Masato Yoshimi[¶]

[§]National Institute of Information and Communications Technology (NICT), Japan.
[¶]TIS Inc., Japan.

February 22, 2022

Abstract

Cache systems are crucial for reducing communication overhead on the Internet. The importance of communication privacy is being increasingly and widely recognized; therefore, we anticipate that nearly all end-to-end communication will be encrypted via secure sockets layer/transport layer security (SSL/TLS) in the near future. Herein we consider a catch-22 situation, wherein the cache server checks whether content has been cached or not, i.e., the cache server needs to observe it, thereby violating end-to-end encryption. We avoid this catch-22 situation by proposing an encrypted cache system which we call Cache-22. To maximize its deployability, we avoid heavy, advanced cryptographic tools, and instead base our Cache-22 system purely on traditional SSL/TLS communication. It employs tags for searching, and its design concept enables the service provider to decide, e.g., via an authentication process, whether or not a particular user should be allowed to access particular content. We provide a prototype implementation of the proposed system using the color-based cooperative cache proposed by Nakajima et al. (IEICE Trans. 2017) under several ciphersuites containing post-quantum key exchanges in addition to ECDHE (Elliptic Curve-based). We consider NIST Post-Quantum Cryptography round 3 finalists and alternate candidates: lattice-based (Kyber, SABER, NTRU), code-based (BIKE), and isogeny-based (SIKE). Compared to direct HTTPS communication between a service provider and a user, employing our Cache-22 system has a merit to drastically reduce communications between a cache server and the service provider (approximately 95%) which is effective in a hierarchical network with a cost disparity.

1 Introduction

1.1 Difficulty in Deploying Services over Encrypted Communication

The importance of communication privacy is being increasingly and widely recognized. For example, NSS labs reported that the amount of secure sockets layer/transport layer security (SSL/TLS) encrypted Internet traffic grew by 90% between July 2015 and July 2016, and predicted that 75% of all the web traffic will be encrypted by 2019 [47]. Actually, Fahmida reported that "*At the start of 2019, 87 percent of Web traffic was encrypted, compared to just 53 percent in 2016.*" [51]. We anticipate that, in the near future, almost all communications will be encrypted. However, although SSL/TLS provides end-to-end encryption, this prevents many useful services from working well.

*An extended abstract appeared at ISITA 2020 [30]. This is the full version.

One such example is virus detection. If a traffic is not encrypted and middlebox can observe the plaintext data, then it has a way to detect whether or not viruses are present. On the contrary, if traffic is encrypted, then inspection becomes much more difficult [20,34,37,56]. Current deep-packet inspection (DPI) [37] solutions involve middleboxes taking a man-in-the-middle (MITM) attack-like approach, i.e., decrypting all traffic, inspecting the plaintext data, and then reencrypting the data. Since this compromises communications privacy, end-to-end encryption cannot meaningfully preserve privacy anymore. Constantin has reported that Kaspersky also employs such MITM-style inspections [24].

1.2 Our Target: A Secure and Highly Deployable End-To-End Encrypted Cache System

Our main aim is to propose a cache system that operates over encrypted communication channels. Cache systems are clearly vital for reducing communication overheads on the Internet, but they raise a dilemma: if they can observe the data, then they compromise privacy. On the other hand, if the data is encrypted, it is non-trivial for the cache server to check whether or not it holds a copy of particular encrypted content. When Moldovan et al. [45] investigated the effectiveness of a web cache in a field study, they noted that traditional caching might become unfeasible over hypertext transfer protocol secure (HTTPS).

In order to establish a secure encrypted cache system, we need to consider how users will decrypt ciphertexts: when the cache server stores ciphertexts that are encrypted by a certain way, it cannot know in advance who will later request them. At first glance, the most reasonable approach is to encrypt the content with user-independent keys, but that means users who request a particular piece of content and receive the corresponding ciphertext from the server will not know the associated decryption key. However, if we attempt to deal with this by encrypting each item with a user-dependent key, then the ciphertext size depends on the number of users who will access that item, leading to problematically large storage-space requirements in practice. One naive solution is, again, to use a MITM-based design. Here, the cache server establishes separate secure channels between (1) itself and the service provider and (2) itself and the end user. It then decrypts the encrypted content (e.g., movies) sent by the provider so it can check whether it has a copy of the content requested by the user. Although both communication channels are encrypted, meaning no third party can observe the content, the server itself is aware of the content it stores and, more seriously, it violates user privacy (e.g., by knowing what movies particular users want to watch).

A promising way to avoid the need for such MITM methods is to employ a cryptographic approach, such as searchable encryption [28] (which allows servers to search encrypted data without decryption), secure data deduplication [57] (which allows servers to check whether pieces of encrypted content are the same), or proxy re-encryption (PRE) [17]. In the latter case, service providers encrypt content using their own public keys, and these ciphertexts are stored on the server. When a user needs to access a piece of encrypted content, the server re-encrypts the ciphertext for the user without having to decrypt it first. Xiong et al. [68] have already proposed a PRE-based system called CloudSeal that provides cache functionality. Wood and Uzun [67] also have proposed a similar PRE-based system in the Content-centric networking (CCN) context. However, under their schemes, the cache server has to manage re-encryption keys whose size depends on the number of users, meaning the system is not scalable in that sense. Attribute-based encryption (ABE) can be employed to solve the storage-space explosion problem above, for example via ABE with constant ciphertext size [13]. Moreover, Lai et al. [39] proposed a bandwidth-efficient encrypted pattern matching protocol for secure middleboxes by employing a variant of symmetric hidden vector encryption. However, the setup costs again raise a significant barrier to deploying

such systems on the Internet. In summary, although employing advanced cryptographic techniques is a promising approach, the associated deployability issues mean it is still better to limit their use as far as possible.

Leguay et al. [40] proposed an elegant approach that they call CryptoCache. With their system, each piece of content is encrypted with a user-independent key k_c to generate the ciphertext C, via $C \leftarrow \mathsf{Enc}_{k_c}(\mathsf{content})$, and the key is also encrypted with a user-dependent key k_i, via $C_{k,i} \leftarrow \mathsf{Enc}_{k_i}(k_c)$. Then, the cache server sends both the ciphertext C and the encrypted decryption key $C_{k,i}$ to the user. Given these, the user can obtain the encryption key k_c, via $k_c \leftarrow \mathsf{Dec}_{k_i}(C_{k,i})$, and hence the content, via $\mathsf{content} \leftarrow \mathsf{Dec}_{k_c}(C)$. The Enc and Dec steps utilize a symmetric key encryption (SKE) scheme, such as AES. One key element of Leguay et al.'s system is that they introduce what they call a pseudo-identifier pid, which we call a tag in our system. Specifically, a tag tag is assigned to each piece of content, chosen independently of it, and the (tag, C) pair is cached. One drawback of CryptoCache is linkability since the same tag is always used for the same content and a pair (tag, C) is sent via a public channel. Leguay et al. also proposed an extension that provides unlinkability, where requests for the same content cannot be detected (except by the (edge) cache server), by additionally employing a public key encryption (PKE) scheme to encrypt the tag together with a temporal key. Since PKE is much less efficient than SKE, in practice it is typically only employed for limited purposes, such as the initial key exchange. Thus, it would be better to avoid using such a scheme where possible, for efficiency reasons. We remark that the PKE part can be removed by employing a secure channel (as in our system) although Leguay et al. did not mention it. However, since C and $C_{k,i}$ are simultaneously sent to the user, unnecessary information is still leaked where $C_{k,i}$ is a ciphertext of a key that can decrypt C. Although we are not sure such additional information leakage detracts the security or not, it would be better to avoid to leak information as much as possible. Moreover, although they discussed the possible implementation of their system over HTTP, they did not provide any concrete implementation result, and thus it is not clear whether their system is feasible in practice or not.

1.3 Our Contribution

In this paper, we propose an encrypted cache system called Cache-22 that does not employ any additional tools beyond SSL/TLS, which yields a highly deployable system. More precisely, we only require an (initial) key exchange protocol, a SKE scheme, and a hash function, all of which have been implemented in SSL/TLS. In Cache-22, we tweak the following procedure. In CryptoCache, a user receives a pair of the encrypted content and the encrypted decryption key $(C, C_{k,i})$ from the cache server. In Cache-22, first, users receive the encrypted content C from the cache server, and later receive the encrypted decryption keys $C_{k,i}$ from the service provider. This design enables the service provider to decide (via an authentication process, for example) whether or not a particular user should be allowed to access particular content. It also allows service providers to control which content users can access. For example, if a user has downloaded several pieces of encrypted content (e.g., several episodes of a drama), the service provider can make some of that content (e.g., the first episode) available for free but require a fee for the rest. Alternatively, the cache server can send the encrypted decryption keys to users alongside the content, and we can select between these options without impacting deployability.

With our Cache-22 system, the cache server manages the encrypted content (ciphertexts) and corresponding tags. Since the tags are chosen independently of the content, the cache server cannot use them to obtain any information about the content.[1] In our implementation, we generate

[1] Andreoletti et al. [9] studied the trade-off between caching and privacy by considering different metrics, such as the hit-rate and retrieval latency. We may employ their results when deciding how to assign tags.

the tags using hash-based message authentication code (HMAC), because it is a pseudorandom function [15]. Our Cache-22 system also provides unlinkability, as only the cache server can detect requests for the same content. Although the CryptoCache extension discussed above also achieves this by employing a PKE scheme, our Cache-22 system does not require any PKE scheme.

We also give a formal security model in a cryptographic manner unlike CryptoCache. Although giving such a model is somewhat overlooked, this provable-security analysis attempt is quite effective to understand what secure means here and what conditions are assumed for achieving the security.[2] Due to our motivation of this paper, we require the cache server, which is modeled as an honest-but-curious adversary (i.e., it always follows the protocol procedure but may try to extract some information) in our security definition, cannot obtain information of contents. Here, we note that the cache server is allowed to know the cache hit ratios (i.e., the popularity of particular content), as this is important for deciding what content to cache.[3] We also give a formal definition for unlinkability which is informally defined in the CryptoCache paper.

We also give a prototype implementation of our Cache-22 system, and show that employing our Cache-22 system has a merit to drastically reduce communications between a cache server and a service provider which is effective in a hierarchical network with a cost disparity. For SSL/TLS part, we first employ two ciphersuites that have been widely used, respectively, for efficiency comparison.

1. `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
2. `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`

As a result, the ciphersuite with ECDSA is more efficient than RSA. We remark that any ciphersuite can be employed since our system is purely generic. Recently, post-quantum ciphersuites have been launched [3], and for post-quantum era it is quite important whether these ciphersuites achieves a similar efficiency compared to those of when we employ classical ciphersuites. Thus, our implementation contains post-quantum key exchanges following (precisely, these are key encapsulation mechanisms (KEMs)). We considered NIST Post-Quantum Cryptography (PQC) round 3 finalists and alternate candidates [6].

3. Lattice-based (MLWE): Kyber512 [18]
4. Lattice-based (MLWR): LightSABER [27]
5. Lattice-based (NTRU): NTRU-HPS-2048-509 [22]
6. Code-based: BIKE1-L1-FO [11]
7. Isogeny-based: SIKE-p434 [1]

where MLWE stands for the module learning with errors problem and MLWR stands for the module learning with rounding problem. NTRU is the name of the cryptosystem and is categorised as a lattice based one in the NIST PQC Standardization document. We choice parameters that are

[2]For example, Cohn-Gordon et al. [23] gave a formal security model of the Signal protocol which has been adopted by WhatsApp, Facebook Messenger, and so on. Aviram et al. [14] also gave a formal security model of forward security for TLS 0-RTT (zero round-trip time).

[3]We may have to consider website fingerprinting attacks [19, 26, 33, 48] where an attacker identifies a user's web browsing information from the metadata information such as packet size, the timing information between packets, and the direction of the packet, without breaking the underlying encryption. If we need to hide the content popularity data to protect user preferences, we could employ Oblivious RAM (ORAM) [58] although this would sacrifice deployability and efficiency. Another approach was investigated by Cui et al. [25] that involves hiding the content popularity data by combining searchable encryption with a multi-CDN strategy, although this requires a performance trade-off.

insisted to have post-quantum security level similar to AES-128. As the concrete ciphersuite, we employed these 5 post-quantum KEMs instead of employing ECDHE above, and other parts are the same as those of `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`. We compare these 7 ciphersuites in total. For implementing PQC ciphersuites, we employed Go library for TLS key exchange [4] and goliboqs [5] which is a Go wrapper around liboqs (C library for quantum-safe cryptography) provided in [3]. We do not employ Classic-McEliece [16] in this paper, although it is one of the NIST PQC round 3 finalists. The reason is that goliboqs, which is employed in our implementation, does not support Classic-McEliece owing to its large public key size. Thus in this paper we employed BIKE, which is a code-based one as in Classic-McEliece and is one of the NIST PQC round 3 alternate candidates. As a remark, we did not consider post-quantum authentication. The reason is, as mentioned by Alkim et al. [8], "*the protection of stored transcripts against future decryption using quantum computers is much more urgent than post-quantum authentication. Authenticity will most likely be achievable in the foreseeable future using proven pre-quantum signatures and attacks on the signature will not compromise previous communication".* Thus, we mainly focus on post-quantum key exchange in this paper.

Together with SSL/TLS, we employed a color-based cooperative cache system [46] as the underlying cache system. This associates servers and caches via color tags using its color tag management algorithm. In addition to being efficient, this system is a good match for Cache-22 as the color tag management algorithm is based purely on the content popularity. In addition, we also have the alternative of assigning colors to tags irrespective of the content popularity. For example, if we have reason to believe a particular piece of content (e.g., a movie or drama) will be popular even before it becomes available, it can be cached in advance. We can also control how likely particular content is to be cached by assigning multiple colors to the tags. Due to these features, we chose to employ a color-based cache system, but other systems could also be used.

1.4 Differences from Proceedings Version

An extended abstract appeared at ISITA 2020 [30]. In this full version, we give a formal syntax and security model (in a cryptographic manner) that we omitted in [30] (Section 2). We also add security analysis section (Section 4). Moreover, we re-consider our implementation environment and ciphersuites (Section 5). Briefly, in [30], we set up six virtual machines (VMs) for the allocated role, on a public cloud provided by GMO [2]. In this version, we prepare on-premises environment, and set up six physical hosts instead to VMs. Moreover, our implementation contains post-quantum ciphersuites that were not contained in [30].

1.5 Related Work

The IBM Mobile First Platform [50] has launched a service called Encrypted Cache which is a mechanism for storing sensitive data on the client side. In other words, the client downloads data and caches it in local encrypted storage. However, since our goal is to provide a cache system for encrypted communications, their system has totally different aims. Varnish Software [65] offers their Varnish Total Encryption service, which prevents accidental cache leaks caused by vulnerabilities such as Heartbleed [60] and Meltdown [42]. However, this is not meant to replace SSL/TLS transport encryption as it encrypts and decrypts the data at rest, meaning that the cache server can observe the data even though it effectively prevents cache leakage. Ericsson [31] has developed an encrypted cache system called Blind cache which takes a somewhat different approach than Cache-22 does. Here, the user (client/browser) obtains the decryption keys first and only later obtains the encrypted content. Consequently, decryption keys, which should generally

be securely maintained, may be given to users unnecessarily. By contrast, our Cache-22 system only gives decryption keys to users when necessary. In addition, as mentioned before, Cache-22 can easily be extended to enable service providers to decide whether or not to send decryption keys to users. Sevilla et al. proposed a system called GroupSec [55], driven by the same concern that motivated us, namely the fact that HTTPS, although it ensures complete security for clients and servers, also interferes with transparent content caching by middleboxes. In addition, they pointed out that previous attempts [43, 44, 49, 52, 61, 62] had modeled middleboxes as trustworthy entities that can and should be authorized to read and write content. To remedy this, GroupSec defines content groups, where members of a group are authorized to view particular content objects. Briefly, each piece of content is encrypted with the service provider's public key, the corresponding decryption key is distributed via HTTPS, and transmission requests and responses are sent via HTTP (not HTTPS). This, as they noted, has the drawback of significantly reducing privacy when compared with HTTPS: users can see whether or not other users in the same group have requested particular content, and can also identify members of groups they do not themselves belong to. In addition, content is encrypted via PKE, and 2048-bit RSA is recommended.[4] By contrast, our Cache-22 system does not define such groups and all communications are separately encrypted via SSL/TLS. Moreover, we only employ public key cryptosystem during the initial key exchange phase (to establish a secure channel). All content is encrypted via SKE, namely AES128, which significantly reduces the computational overhead compared with employing PKE to encrypt/decrypt content. Araldo et al. [12] an architecture that they call Stochastic Dynamic Cache Partitioning, where the Internet service provider partitions the cache space into slices, assigns each slice to a different content provider, and then allows them to manage their slices remotely. This architecture enables the transparent caching of encrypted content. Andreoletti et al. [10] proposed a secret sharing-based cache system that allows the Internet service provider to calculate the amount of cache storage each content provider is entitled to receive while guaranteeing network neutrality and resource efficiency. Usman [64] also proposed an encrypted cache system by employing the convergent encryption [29], where a content is hashed and it is used as encryption key such as $\mathsf{Enc}_k(\mathsf{content})$ and $k = \mathsf{Hash}(\mathsf{content})$, and the key k is also encrypted by the public keys of all authorized readers of the file. Jensen [38] proposed a cache system CryptoCache as independent to the Leguay et al. work [40]. In the system, a mobile device (client) is required to have sufficient capacity to store file encryption keys, e.g., 256 RSA key pairs are recommended in the paper, and each file is encrypted by a PKE scheme. Again, we do not employ any public key cryptosystem for encryption.

Providing post-quantum security is now drawing attention, and many implementation results with post-quantum security have been shown so far. For example, TLS with post-quantum security [54], end-to-end post quantum encryption [63], and an initial secret key sharing protocol for Signal [32] and so on.

2 Syntax and Security Model

In this section, we define the syntax of Cache-22, and give a formal security definition. In our system, there are three entities, a service provider (SP), a cache server (CS), and a user. Our solution is simple-but-effective. The design concept is that we pay attention to the size of contents

[4] As another problem, MD5 hash function is recommended in [55] although it has been broken [66]. It should be replaced to other hash functions. Since several attacks for SHA-1 also have been reported [41, 59], selecting SHA-2 or SHA-3 are adequate options. We employ SHA-256 (which is a SHA-2 family with 256-bit output) in our implementation.

(e.g., movies) are much bigger than that of keys for encryption/decryption. Thus, we consider to minimize the number of communications for encrypted contents. Although we require some interactions for sending keys, we can employ TLS session resumption [53], and no further key exchange protocol is required except the initial handshake phase.

2.1 Syntax of Encrypted Cache System

Our system consists of three algorithms and three interactive sub-protocols. For interactive ones, we denote

$$(\mathsf{out}_1, \mathsf{out}_2) \leftarrow \mathsf{Protocol}(\mathsf{Entity}_1(\mathsf{input}_1), \mathsf{Entity}_2(\mathsf{input}_2))$$

where $\mathsf{Protocol}$ is run between Entity_1 (which takes as input input_1) and Entity_2 (which takes as input input_2), and $\mathsf{Protocol}$ is initially run by Entity_1. Finally, Entity_1 (resp. Entity_2) obtains out_1 (resp. out_2) as the output of $\mathsf{Protocol}$. We remark that the output may be ϵ when an entity obtains nothing. For the sake of simplicity, basically we use the two-entity notation but sometimes a third entity appears in the protocol, e.g., the $\mathsf{SendContent}$ protocol is run by a user and CS but CS sends some values to SP as an internal process of the protocol. We assume that there are n contents. But without loss of generality, SP always extends the number of contents by running the $\mathsf{GenTable}$ algorithm again. We also assume that there is a SKE scheme $(\mathsf{Enc}, \mathsf{Dec})$ where for a key $k \in \mathcal{K}$ and a message $M \in \mathcal{M}$, $\mathsf{Dec}_k(C) = M$ holds where $C \leftarrow \mathsf{Enc}_k(M)$, $\mathcal{K}$ is a key space, and $\mathcal{M}$ is a message space. We also assume that a user knows a content name $\mathsf{c\_name}$, and SP can decide the corresponding content $\mathsf{content}_i \in \mathcal{M}$ from $\mathsf{c\_name}$. Let $\mathsf{CacheTbl}$ be the cache table managed by CS which has the structure $\mathsf{CacheTbl} = \{(\mathsf{tag}_i, C_i)\}$, and is initiated as $\emptyset$. Although we simply denote $\mathsf{CacheTbl} = \{(\mathsf{tag}_i, C_i)\}$ here, we can employ any cache systems.

Definition 2.1 (Syntax of Cache-22)**.** *An encrypted cache system* $\mathsf{Cache\text{-}22}$ *consists of the following three PPT (probabilistic polynomial-time) algorithms and three interactive protocols:*

- $\mathsf{GenTable}(1^\kappa, 1^\lambda, \mathsf{SetOfContents})$*: The table generation algorithm run by SP takes as input security parameters* $\kappa, \lambda \in \mathbb{N}$ *and a set of contents* $\mathsf{SetOfContents} = \{\mathsf{content}_i\}_{i=1}^n$*.* $\mathcal{K} := \{0,1\}^\lambda$*, and* $\mathcal{M}$ *is implicitly defined by* λ*. For each* $\mathsf{content}_i \in \mathcal{M}$*, randomly choose* $k_{c,i} \leftarrow \mathcal{K}$ *and* $\mathsf{tag}_i \leftarrow \mathcal{TAG}$ *where* $\mathcal{TAG} := \{0,1\}^\kappa$ *is a tag space, and encrypt* $\mathsf{content}_i$ *such that* $C_i \leftarrow \mathsf{Enc}_{k_{c,i}}(\mathsf{content}_i)$*. Output a content table* $\mathsf{CTbl} = \{(\mathsf{content}_i, \mathsf{tag}_i, C_i, k_{c,i})\}$*.*

- $\mathsf{ContentRequest}(\mathsf{User}(\mathsf{c\_name}, ID), \mathsf{SP}(\mathsf{ConTbl}))$*: The* $\mathsf{ContentRequest}$ *protocol between a user and SP takes as input a content name* $\mathsf{c\_name}$ *and the user identity* ID *from the user,*[5] *and takes as input* ConTbl *from SP. The protocol is run as follows. (1) The user sends* $(\mathsf{c\_name}, ID)$ *to SP. (2) SP decides* $\mathsf{content}_i$ *from* $\mathsf{c\_name}$*, and retrieves the corresponding* $(\mathsf{content}_i, \mathsf{tag}_i, C_i, k_{c,i})$ *from* ConTbl*. SP sends* tag_i *to the user.*

- $\mathsf{SendContent}(\mathsf{User}(\mathsf{tag}_i, ID), \mathsf{CS}(\mathsf{CacheTbl}))$*: The content sending protocol between a user and CS takes as input* (tag_i, ID) *from the user, and takes as input* $\mathsf{CacheTbl}$ *from CS. The protocol is run as follows. (1) The user sends a request* (tag_i, ID) *to CS. (2) CS checks whether* tag_i *is preserved on* $\mathsf{CacheTbl}$*. If yes, CS retrieves* (tag_i, C_i) *from* $\mathsf{CacheTbl}$ *by using* tag_i*, sends* C_i *to the user, and sends* (tag_i, ID) *to SP. If no, CS runs the* $\mathsf{CacheRequest}$ *protocol with SP (which is defined later), obtains* C_i*, preserves* (tag_i, C_i) *to* $\mathsf{CacheTbl}$*, and sends* C_i *to the user.*

[5] Alternatively, SP can choose a temporal identity and give it to the user.

- $\mathsf{CacheRequest}(\mathsf{CS}(\mathsf{tag}_i, ID), \mathsf{SP}(\mathsf{ConTbl}))$*: The cache request protocol between CS and SP takes as input* (tag_i, ID) *from CS, and takes as input* ConTbl *from SP. The protocol is run as follows. (1) CS sends* (tag_i, ID) *to SP. (2) SP retrieves* $(\mathsf{content}_i, \mathsf{tag}_i, C_i, k_{c,i})$ *from* ConTbl *by using* tag_i*, and sends* C_i *to CS.*

- $\mathsf{SendKey}(ID, k_{c,i})$*: The key sending algorithm run by SP takes as input* $(ID, k_{c,i})$*. Send* $k_{c,i}$ *to the user whose identity is* ID*.*

- $\mathsf{ObtainContent}(\mathsf{tag}_i, C_i, k_{c,i})$*: The content obtaining algorithm run by a user takes as input* $(\mathsf{tag}_i, C_i, k_{c,i})$*. Output* $\mathsf{content}_i \leftarrow \mathsf{Dec}_{k_{c,i}}(C_i)$.

The correctness is defined as follows. It guarantees that if all algorithms are honestly run, then a user always can obtain the content that the user requests. For all $\mathsf{ConTbl} \leftarrow \mathsf{GenTable}(1^\kappa, 1^\lambda, \mathsf{SetOfContents})$, all $\mathsf{content}_i \in \mathcal{M}$, $k_{c,i} \leftarrow \mathcal{K}$, and $\mathsf{tag}_i \leftarrow \mathcal{TAG}$, for $C_i \leftarrow \mathsf{Enc}_{k_{c,i}}(\mathsf{content}_i)$, where $\mathsf{ConTbl} = \{(\mathsf{content}_i, \mathsf{tag}_i, C_i, k_{c,i})\}$, and for $(\mathsf{tag}_j, C_j) \in \mathsf{CacheTbl}$, $\Pr[\mathsf{content}_i \leftarrow \mathsf{Dec}_{k_{c,i}}(C_i)] = 1$ holds, and if $\mathsf{tag}_i = \mathsf{tag}_j$ then $C_i = C_j$ holds. We remark that the syntax and the correctness do not say anything about communication privacy, and it is defined in the next subsection.

The actual flow of our system is described as follows. Here, we assume that SP has run $\mathsf{ConTbl} \leftarrow \mathsf{GenTable}(1^\kappa, 1^\lambda, \mathsf{SetOfContents})$ in the initial phase.

1. A user and SP run $(\mathsf{tag}_i, \epsilon) \leftarrow \mathsf{ContentRequest}(\mathsf{User}(\mathsf{c\_name}, ID), \mathsf{SP}(\mathsf{ConTbl}))$.
2. The user and CS run $(C_i, \epsilon) \leftarrow \mathsf{SendContent}(\mathsf{User}(\mathsf{tag}_i, ID), \mathsf{CS}(\mathsf{CacheTbl}))$. We remark that CS may run $(C_i, \epsilon) \leftarrow \mathsf{CacheRequest}(\mathsf{CS}(\mathsf{tag}_i, ID), \mathsf{SP}(\mathsf{ConTbl}))$ with SP internally if tag_i is not cached.
3. SP runs $\mathsf{SendKey}(ID, k_{c,i})$ and the user obtains $k_{c,i}$.
4. The user runs $\mathsf{content}_i \leftarrow \mathsf{ObtainContent}(\mathsf{tag}_i, C_i, k_{c,i}))$.

2.2 Security Model

Next, we define the security of the encrypted cache system. Due to our motivation of this paper, we require that no CS can obtain information of contents. Thus, CS is modeled as an honest-but-curious adversary (i.e., it always follows the protocol procedure but may try to extract some information) in our security definition.

The formal model is defined as follows. An adversary $\mathcal{A}$ is modeled as CS. If $\mathcal{A}$ colludes with SP, then $\mathcal{A}$ can obtain all contents, all encryption/decryption keys $k_{c,i}$, and all secret values maintained by SP (that include all SSL/TLS session keys shared between SP and users). Thus, as a realistic restriction, we assume that $\mathcal{A}$ does not collude with SP. Instead, $\mathcal{A}$ is allowed to corrupt some users, and then $\mathcal{A}$ can obtains all values that these users have. Intuitively, the following security model guarantees that no $\mathcal{A}$ can obtain information of content from tag and ciphertext pair (tag^*, C^*) if corrupted users do not obtain k_c^* that can decrypt C^*. In other words, $\mathcal{A}$ cannot obtain information of content from requests sent from honest users. We adopt the standard indistinguishability-based notion which guarantees that $\mathcal{A}$ cannot distinguish whether (tag^*, C^*) is an encryption of $\mathsf{content}_0^*$ or $\mathsf{content}_1^*$ even $\mathcal{A}$ can select contents.

Definition 2.2 (Secure Encrypted Cache System)**.**

Setup. *At the outset of the game, the challenger runs* $\mathsf{ConTbl} \leftarrow \mathsf{GenTable}(1^\kappa, 1^\lambda, \mathsf{SetOfContents})$*, and initializes lists* $\mathsf{HU} := \{ID_1, \ldots, ID_m\}$ *where* $m \in \mathbb{N}$ *is the number of users,* $\mathsf{CU} := \emptyset$*, and*

$\mathsf{CR} := \emptyset$ *which stands for honest users list, corrupted users list, and content request list, respectively. The challenger initializes* $\mathsf{ind} = 0$*. The challenger further picks a random bit* $b \leftarrow \{0,1\}$ *and keeps it secret.*

Queries. $\mathcal{A}$ *can adaptively make the following five types of queries to the challenger in arbitrary order:* Corruption*,* ContentRequest*,* SendContent*,* SendKey*, and* Challenge *queries* $\mathcal{A}$ *can query the first four arbitrarily polynomially many times and the fifth only once.*

- Corruption *Query: If* $\mathcal{A}$ *submits* $ID \in \mathsf{HU}$ *to the challenger, then the challenger updates* $\mathsf{HU} \leftarrow \mathsf{HU} \setminus \{ID\}$ *and* $\mathsf{CU} \leftarrow \mathsf{CU} \cup \{ID\}$*.*
- ContentRequest *Query: If* $\mathcal{A}$ *submits* ID *to the challenger, then the challenger chooses* $\mathsf{c\_name}$*, and runs* $\mathsf{ContentRequest}(\mathsf{User}(\mathsf{c\_name}, ID), \mathsf{SP}(\mathsf{ConTbl}))$*. We remark that if* $ID \in \mathsf{CU}$*, then* $\mathcal{A}$ *can observe all communications between SP and the user* ID *in a plaintext manner (in this case,* $\mathcal{A}$ *can know* $\mathsf{c\_name}$ *and the corresponding tag* tag*). Finally, the challenger updates* $\mathsf{CR} \leftarrow \mathsf{CR} \cup \{(\mathsf{tag}, C, ID, \mathsf{ind}, k_c)\}$ *where* $\mathsf{ind} \in \mathbb{N}$ *indicates the number of* ContentRequest *queries and* k_c *is the encryption key used in the encryption of the content, and updates* $\mathsf{ind} \leftarrow \mathsf{ind} + 1$*.*
- SendContent *Query: If* $\mathcal{A}$ *submits* (ID, ind) *to the challenger, then the challenger returns* $\perp$ *if* $(\mathsf{tag}, \cdot, ID, \mathsf{ind}, \cdot)$ *is not contained in* CR*. Otherwise, the challenger retrieves the entry* $(\mathsf{tag}, C, ID, \mathsf{ind}, k_c)$*, and runs* $\mathsf{SendContent}(\mathsf{User}(\mathsf{tag}, ID), \mathsf{CS}(\mathsf{CacheTbl}))$*. Then,* $\mathcal{A}$ *behaves as CS. We remark that* $\mathcal{A}$ *can observe* tag *regardless of* $ID \in \mathsf{CU}$ *or* $ID \in \mathsf{HU}$ *since the challenger sends* tag *to* $\mathcal{A}$*.*
- SendKey *Query: If* $\mathcal{A}$ *submits* (ID, ind) *to the challenger, then the challenger returns* $\perp$ *if* $(\cdot, \cdot, ID, \mathsf{ind}, \cdot)$ *is not contained in* CR*. Otherwise, the challenger retrieves the entry* $(\mathsf{tag}, C, ID, \mathsf{ind}, k_c)$*, and runs* $\mathsf{SendKey}(ID, k_c)$*. Again, if* $ID \in \mathsf{CU}$*, then* $\mathcal{A}$ *can observe all communications between SP and the user* ID *in a plaintext manner (in this case,* $\mathcal{A}$ *can know* k_c*).*
- Challenge *Query: If* $\mathcal{A}$ *submits* ID^* *and two content names* $\mathsf{c\_name}_0^*$ *and* $\mathsf{c\_name}_1^*$ *to the challenger, where the size of* $\mathsf{content}_0^*$ *and that of* $\mathsf{content}_1^*$ *is the same, if* $ID^* \in \mathsf{CU}$*, then the challenger returns* $\perp$*, outputs a random bit, and aborts. Otherwise (i.e.,* $ID^* \in \mathsf{HU}$*), the challenger retrieves* $(\mathsf{content}_b^*, \mathsf{tag}^*, C^*, k_c^*)$ *from* ConTbl *by using* $\mathsf{c\_name}_b^*$*. If* $\mathcal{A}$ *has obtain* k_c^**, then the challenger returns* $\perp$*, outputs a random bit, and aborts. Otherwise, the challenger runs* $(\mathsf{tag}^*, \epsilon) \leftarrow \mathsf{ContentRequest}(\mathsf{User}(\mathsf{c\_name}_b^*, ID^*), \mathsf{SP}(\mathsf{ConTbl}))$*, and runs* $\mathsf{SendContent}(\mathsf{User}(\mathsf{tag}^*, ID^*), \mathsf{CS}(\mathsf{CacheTbl}))$ *with* $\mathcal{A}$ *as CS. If* (tag^*, C^*) *is not cached, then* $\mathcal{A}$ *runs* $\mathsf{CacheRequest}(\mathsf{CS}(\mathsf{tag}^*, ID^*), \mathsf{SP}(\mathsf{ConTbl}))$ *with the challenger as SP. Finally,* $\mathcal{A}$ *obtains* (tag^*, C^*)*. Hereafter,* $\mathcal{A}$ *is not allowed to obtain* k_c^* *via* SendKey *queries, and is not allowed to corrupt* ID^**.*

Guess. $\mathcal{A}$ *outputs a guess* $b' \in \{0,1\}$ *for* b*. We say that* Cache-22 *is secure if the advantage*

$$\mathsf{Adv}^{\mathsf{secure}}_{\mathsf{Cache\text{-}22},\mathcal{A}}(\kappa, \lambda) = |\Pr[b = b'] - 1/2|$$

is negligible for any PPT adversary $\mathcal{A}$*.*

We remark that $\mathcal{A}$ (i.e., CS) can know cache hit ratios by observing which tags are requested by users. This information shows popularity of contents, and we can use it for deciding what contents should be cached, as in [55]. Thus, we accept that $\mathcal{A}$ knows cache hit ratios.

Next, we define unlinkability in a formal way. Leguay et al. [40] informally defined unlinkability as: requests for the same content cannot be detected except by the cache server. That is, the adversary $\mathcal{A}$ is modeled as outsider or users. More precisely, $\mathcal{A}$ is allowed to observe all communications (transcripts of the protocols) among users, CS, and SP. Moreover, $\mathcal{A}$ is allowed to corrupt users, and then $\mathcal{A}$ can observe all communications between the users and either CS or SP in a plaintext manner, and can obtain all values that they have. Intuitively, the following security model guarantees that no $\mathcal{A}$ can distinguish whether a content has been requested before or not if the user that requests the content is not corrupted. In other words, even if $\mathcal{A}$ has observed protocol transcripts for $(\mathsf{c\_name}_0^*, ID)$ where $ID \in \mathsf{HU}$, $\mathcal{A}$ cannot decide whether or not protocols for $(\mathsf{c\_name}_0^*, ID^*)$ are run in the challenge phase.

Definition 2.3 (Unlinkability)**.**

Setup. *At the outset of the game, the challenger runs* $\mathsf{ConTbl} \leftarrow \mathsf{GenTable}(1^\kappa, 1^\lambda, \mathsf{SetOfContents})$*, and initializes lists* $\mathsf{HU} := \{ID_1, \ldots, ID_m\}$ *where* $m \in \mathbb{N}$ *is the number of users,* $\mathsf{CU} := \emptyset$*, and* $\mathsf{CR} := \emptyset$ *which stands for honest users list, corrupted users list, and content request list, respectively. The challenger initializes* $\mathsf{ind} = 0$*. The challenger further picks a random bit* $b \leftarrow \{0, 1\}$ *and keeps it secret.*

Queries. $\mathcal{A}$ *can adaptively make the following five types of queries to the challenger in arbitrary order:* Corruption*,* ContentRequest*,* SendContent*,* SendKey*, and* Challenge *queries* $\mathcal{A}$ *can query the first four arbitrarily polynomially many times and the fifth only once.*

- Corruption *Query: If* $\mathcal{A}$ *submits* $ID \in \mathsf{HU}$ *to the challenger, then the challenger updates* $\mathsf{HU} \leftarrow \mathsf{HU} \setminus \{ID\}$ *and* $\mathsf{CU} \leftarrow \mathsf{CU} \cup \{ID\}$*.*

- ContentRequest *Query: If* $\mathcal{A}$ *submits* $(\mathsf{c\_name}, ID)$ *to the challenger, then the challenger runs* $\mathsf{ContentRequest}(\mathsf{User}(\mathsf{c\_name}, ID), \mathsf{SP}(\mathsf{ConTbl}))$*.* $\mathcal{A}$ *is allowed to observe the protocol transcript. We remark that if* $ID \in \mathsf{CU}$*, then* $\mathcal{A}$ *can observe all communications between SP and the user* ID *in a plaintext manner (in this case,* $\mathcal{A}$ *can know the corresponding tag* tag*). Finally, the challenger updates* $\mathsf{CR} \leftarrow \mathsf{CR} \cup \{(\mathsf{tag}, C, ID, \mathsf{ind}, k_c)\}$ *where* $\mathsf{ind} \in \mathbb{N}$ *indicates the number of* ContentRequest *queries and* k_c *is the encryption key used in the encryption of the content, and updates* $\mathsf{ind} \leftarrow \mathsf{ind} + 1$*.*

- SendContent *Query: If* $\mathcal{A}$ *submits* (ID, ind) *to the challenger, then the challenger returns* $\perp$ *if* $(\mathsf{tag}, \cdot, ID, \mathsf{ind}, \cdot)$ *is not contained in* CR*. Otherwise, the challenger retrieves the entry* $(\mathsf{tag}, C, ID, \mathsf{ind}, k_c)$*, and runs* $\mathsf{SendContent}(\mathsf{User}(\mathsf{tag}, ID), \mathsf{CS}(\mathsf{CacheTbl}))$*. We remark that if* $ID \in \mathsf{CU}$*, then* $\mathcal{A}$ *can observe all communications between CS and the user* ID *in a plaintext manner (in this case,* $\mathcal{A}$ *can obtain* (tag, C)*).*

- SendKey *Query: If* $\mathcal{A}$ *submits* (ID, ind) *to the challenger, then the challenger returns* $\perp$ *if* $(\cdot, \cdot, ID, \mathsf{ind}, \cdot)$ *is not contained in* CR*. Otherwise, the challenger retrieves the entry* $(\mathsf{tag}, C, ID, \mathsf{ind}, k_c)$*, and runs* $\mathsf{SendKey}(ID, k_c)$*. Again, if* $ID \in \mathsf{CU}$*, then* $\mathcal{A}$ *can observe all communications between SP and the user* ID *in a plaintext manner (in this case,* $\mathcal{A}$ *can know* k_c*).*

- Challenge *Query: Let* $\mathcal{A}$ *submit* ID^* *and a content name* $\mathsf{c\_name}_0^*$*. If* $\mathcal{A}$ *has sent* $(\mathsf{c\_name}_0^*, ID)$ *and* $ID \in \mathsf{CU}$*, then the challenger returns* $\perp$*, outputs a random bit, and aborts. If* $ID^* \in \mathsf{CU}$*, then the challenger returns* $\perp$*, outputs a random bit, and aborts. Otherwise (i.e.,* $ID^* \in \mathsf{HU}$*), the challenger chooses* $\mathsf{c\_name}_1^*$ *that has not been sent as a* ContentRequest *query, and the size of* $\mathsf{content}_0^*$ *and that of* $\mathsf{content}_1^*$ *is the same, and retrieves* $(\mathsf{content}^*, \mathsf{tag}^*, C^*, k_c^*)$ *from* ConTbl *by* $\mathsf{c\_name}_b^*$*. Otherwise, the challenger runs* $(\mathsf{tag}^*, \epsilon) \leftarrow \mathsf{ContentRequest}(\mathsf{User}(\mathsf{c\_name}_\mathsf{b}^*,$

$ID^*), \mathsf{SP}(\mathsf{ConTbl}))$, *and runs* $\mathsf{SendContent}(\mathsf{User}(\mathsf{tag}^*, ID^*), \mathsf{CS}(\mathsf{CacheTbl}))$. *If* (tag^*, C^*) *is not cached, then* $\mathcal{A}$ *runs* $\mathsf{CacheRequest}(\mathsf{CS}(\mathsf{tag}^*, ID^*), \mathsf{SP}(\mathsf{ConTbl}))$. *Hereafter,* $\mathcal{A}$ *is not allowed to send* ContentRequest *queries* $(\mathsf{c\_name}_0^*, ID)$ *for any* $ID \in \mathsf{CU}$. *Moreover,* $\mathcal{A}$ *is not allowed to corrupt* $ID \in \mathsf{HU}$ *if* $(\mathsf{c\_name}_0^*, ID)$ *has been sent as a* ContentRequest *query.*

Guess. $\mathcal{A}$ *outputs a guess* $b' \in \{0,1\}$ *for* b. *We say that* Cache-22 *is unlinkable if the advantage*

$$\mathsf{Adv}^{\mathsf{unlink}}_{\mathsf{Cache\text{-}22},\mathcal{A}}(\kappa, \lambda) = |\Pr[b = b'] - 1/2|$$

is negligible for any PPT adversary $\mathcal{A}$.

3 Proposed System

In this section, we give our Cache-22 system. In our system, there are three entities, a service provider (SP), a cache server (CS), and a user. The design concept is that we pay attention to the size of contents (e.g., movies) are much bigger than that of keys for encryption/decryption. Thus, we consider to minimize the number of communications for encrypted contents. Although we require some interactions for sending keys, we can employ TLS session resumption [53], and no further key exchange protocol is required except the initial handshake phase.

We assume that all communications among a user, CS, and SP are encrypted via SSL/TLS. We explicitly denote the session key as $k_{\mathsf{user}\leftrightarrow\mathsf{CS}}$, $k_{\mathsf{CS}\leftrightarrow\mathsf{SP}}$, and $k_{\mathsf{user}\leftrightarrow\mathsf{SP}}$, respectively. This setting is natural since we expect that almost all communications will be encrypted in the near future. We employ HMAC for generating tags, and thus SP has a secret key for HMAC denoted by k_{hmac} (which is assumed to be generated in advance). Since HMAC is known as a pseudorandom function [15], we assume that tag collision happens with negligible probability, i.e., for (tag_i, C_i), (tag_j, C_j), and $C_i \neq C_j$, $\mathsf{tag}_i \neq \mathsf{tag}_j$ holds with overwhelming probability. Moreover, we use the upper-order 128 bits of tag as the initial vector (IV) for AES-GCM [35]. Then, IV is not re-used for other encryption since tag is pseudorandom. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a IND-CPA secure SKE scheme where for a key $k \in \mathcal{K}$ and a message $M \in \mathcal{M}$, $\mathsf{Dec}_k(C) = M$ holds where $C \leftarrow \mathsf{Enc}_k(M)$, $\mathcal{K}$ is a key space, and $\mathcal{M}$ is a message space. We explicitly indicate IV such that $C_i \leftarrow \mathsf{Enc}_{k_{c,i}}(IV, \mathsf{content}_i)$ and $\mathsf{content}_i \leftarrow \mathsf{Dec}_{k_{c,i}}(IV, C_i)$.

The concrete system is described as follows. Figure 1 shows a flow of Cache-22 system. We assume that there are n contents. But without loss of generality, SP always extends the number of contents. We also assume that a user knows a content name c_name, and SP can decide the corresponding content $\mathsf{content}_i \in \mathcal{M}$ from c_name. Let CacheTbl be the cache table managed by CS which has the structure $\mathsf{CacheTbl} = \{(\mathsf{tag}_i, C_i)\}$, and is initiated as $\emptyset$. Although we simply denote $\mathsf{CacheTbl} = \{(\mathsf{tag}_i, C_i)\}$ here, we can employ any cache systems.

- $\mathsf{GenTable}(1^\kappa, 1^\lambda, \mathsf{SetOfContents})$: The table generation algorithm run by SP takes as input security parameters $\kappa, \lambda \in \mathbb{N}$ and a set of contents $\mathsf{SetOfContents} = \{\mathsf{content}_i\}_{i=1}^n$. $\mathcal{K} := \{0,1\}^\lambda$, and $\mathcal{M}$ is implicitly defined by λ. For each $\mathsf{content}_i \in \mathcal{M}$, randomly choose $k_{c,i} \leftarrow \mathcal{K}$ and $\mathsf{tag}_i \leftarrow \mathsf{HMAC}_{k_{\mathsf{hmac}}}(\mathsf{content}_i)$, retrieve IV from tag_i, and encrypt $\mathsf{content}_i$ such that $C_i \leftarrow \mathsf{Enc}_{k_{c,i}}(IV, \mathsf{content}_i)$. Output a table $\mathsf{ConTbl} = \{(\mathsf{content}_i, \mathsf{tag}_i, C_i, k_{c,i})\}$.
- $\mathsf{ContentRequest}(\mathsf{User}(\mathsf{c\_name}, ID), \mathsf{SP}(\mathsf{ConTbl}))$: The ContentRequest protocol between a user and SP takes as input a content name c_name and the user identity ID from the user, and takes as input ConTbl from SP. The protocol is run as follows. (1) The user sends (c_name, ID) to SP via the secure channel generated by $k_{\mathsf{user}\leftrightarrow\mathsf{SP}}$. (2) SP decides $\mathsf{content}_i$ from c_name,

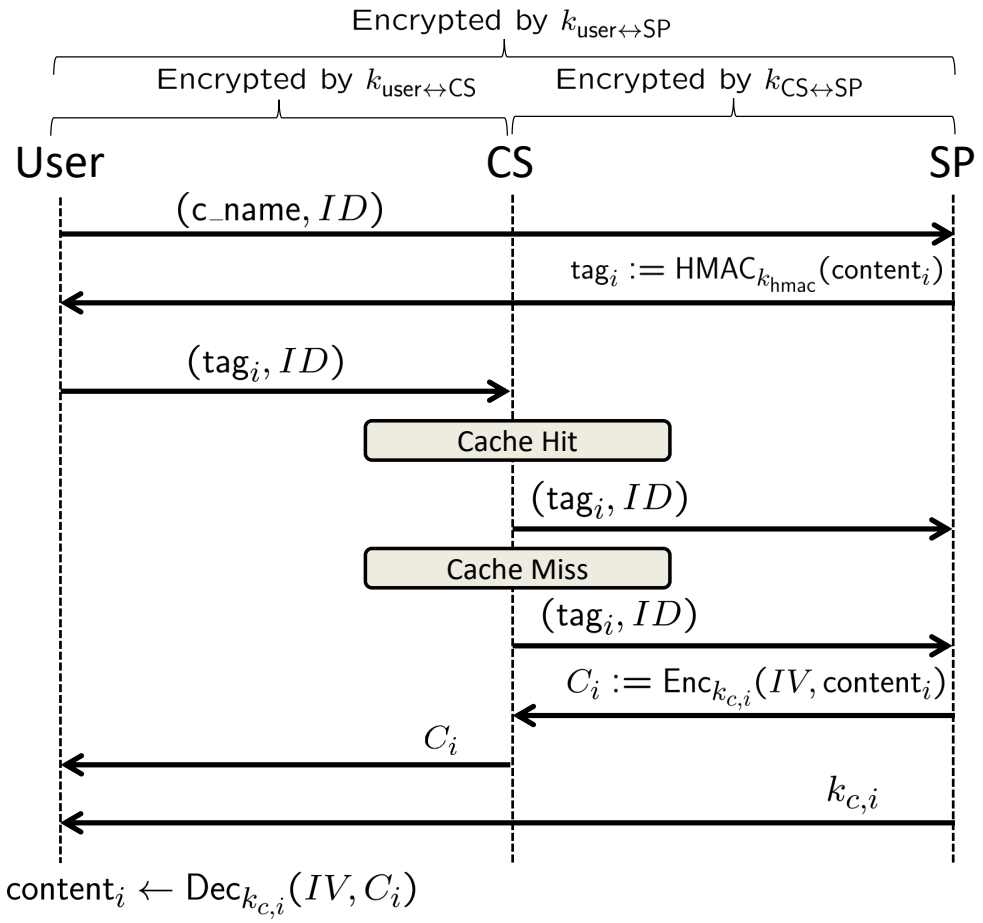


Figure 1: Flow of Our Cache-22 System

and retrieves the corresponding $(\mathsf{content}_i, \mathsf{tag}_i, C_i, k_{c,i})$ from ConTbl. SP sends tag_i to the user via the secure channel.

- $\mathsf{SendContent}(\mathsf{User}(\mathsf{tag}_i, ID), \mathsf{CS}(\mathsf{CacheTbl}))$: The content sending protocol between a user and CS takes as input (tag_i, ID) from the user, and takes as input $\mathsf{CacheTbl}$ from CS. The protocol is run as follows. (1)

- The user sends a request (tag_i, ID) to CS via the secure channel generated by $k_{\mathsf{user}\leftrightarrow\mathsf{CS}}$. (2) CS checks whether tag_i is preserved on $\mathsf{CacheTbl}$. If yes, CS retrieves (tag_i, C_i) from $\mathsf{CacheTbl}$ by using tag_i, sends C_i to the user via the secure channel, and sends (tag_i, ID) to SP via the secure channel generated by $k_{\mathsf{CS}\leftrightarrow\mathsf{SP}}$. If no, CS runs the $\mathsf{CacheRequest}$ protocol with SP (which is defined later), obtains C_i, preserves (tag_i, C_i) to $\mathsf{CacheTbl}$, and sends C_i to the user via the secure channel.

- $\mathsf{CacheRequest}(\mathsf{CS}(\mathsf{tag}_i, ID), \mathsf{SP}(\mathsf{ConTbl}))$: The cache request protocol between CS and SP takes as input (tag_i, ID) from CS, and takes as input ConTbl from SP. The protocol is run as follows. (1)

- CS sends (tag_i, ID) to SP via the secure channel generated by $k_{\mathsf{CS}\leftrightarrow\mathsf{SP}}$. (2) SP retrieves $(\mathsf{content}_i, \mathsf{tag}_i, C_i, k_{c,i})$ from ConTbl by using tag_i, and sends C_i to CS via the secure channel.

- $\mathsf{SendKey}(ID, k_{c,i})$: The key sending algorithm run by SP takes as input $(ID, k_{c,i})$. Send $k_{c,i}$ to the user whose identity is ID via the secure channel generated by $k_{\mathsf{user}\leftrightarrow\mathsf{SP}}$.
- $\mathsf{ObtainContent}(\mathsf{tag}_i, C_i, k_{c,i})$): The content obtaining algorithm run by a user takes as input $(\mathsf{tag}_i, C_i, k_{c,i})$. Retrieve IV from tag_i. Output $\mathsf{content}_i \leftarrow \mathsf{Dec}_{k_{c,i}}(IV, C_i)$.

4 Security

Obviously, Cache-22 is correct if tag collision does not happen and IV is not re-used for other encryption. More concretely, we need to assume that the underlying SKE scheme is also correct, i.e., for all $M \in \mathcal{M}$, $k \in \mathcal{K}$, and initial vector IV, $\Pr[M \leftarrow \mathsf{Dec}_k(IV, \mathsf{Enc}_k(IV, M))] = 1$. Moreover, Cache-22 is secure in the sense of Definition 2.2. That is, due to the pseudorandomness of HMAC, tags are independently chosen from contents. Thus, thanks to the IND-CPA security of the SKE scheme, information of b is not revealed from C^*. In other words, if $\mathcal{A}$ breaks the security, we can construct an algorithm that breaks the IND-CPA security of the SKE scheme. Moreover, Cache-22 is unlinkable in the sense of Definition 2.3. If $\mathcal{A}$ sends $\mathsf{ContentRequest}$ queries ($\mathsf{c\_name}_0^*$, ID^*), then ID^* is not corrucpted due to the rule of the security game. Thus, $\mathcal{A}$ cannot obtain the pair of the corresponding tag and ciphertext since it is sent via the secure channel and $\mathcal{A}$ does not have the corresponding shared key. Then, thanks to the IND-CPA security of the SKE scheme, information of b is not revealed from C^*. As a possible case, even though $\mathcal{A}$ does not know $\mathsf{c\_name}_1^*$, $\mathcal{A}$ may send $(\mathsf{c\_name}_1^*, ID)$ as a $\mathsf{ContentRequest}$ query where $ID \in \mathsf{CU}$ after the challenge phase. Then, $\mathcal{A}$ can break unlinkability as follows: Let assume that $\mathcal{A}$ does not send a $\mathsf{ContentRequest}$ query $(\mathsf{c\_name}_0^*, ID^*)$. Then, $\mathcal{A}$ is allowed to corrupt ID^*. Then, $\mathcal{A}$ can obtain (tag^*, C^*) in the challenge phase. By comparing tags, $\mathcal{A}$ can decide whether C^* is a ciphertext of $\mathsf{content}_0^*$ or $\mathsf{content}_1^*$. Thus, we need to assume that the name space of contents is sufficiently big, and assume that the probability of sending the query is negligible. In other words, if $\mathcal{A}$ breaks the unlinkability under the assumption, then we can construct an algorithm that breaks the IND-CPA security of the SKE scheme.

5 Prototype Implementation

In this section, we give our implementation results. In our implementation, we set $\kappa = 256$ and $\lambda = 128$ which ensure that tag collision happens with negligible probability 2^{-128}, and the underlying SKE scheme is 128-bit secure. Concretely, we employ $\mathsf{AES\text{-}GCM}$ as the underlying SKE scheme. For SSL/TLS part, we first employ two ciphersuites that have been widely used, respectively, for efficiency comparison. As a result, we can say that our systems efficient when ECDSA is employed compared to the case that RSA is employed.

1. `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
2. `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`

Next, we compare post-quantum ciphersuites. We employed these 5 post-quantum KEMs (Kyber512, LightSABER, NTRU-HPS-2048-509, BIKE1-L1-FO, SIKE-p434) instead of employing ECDHE above, and other parts are the same as those of `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`.

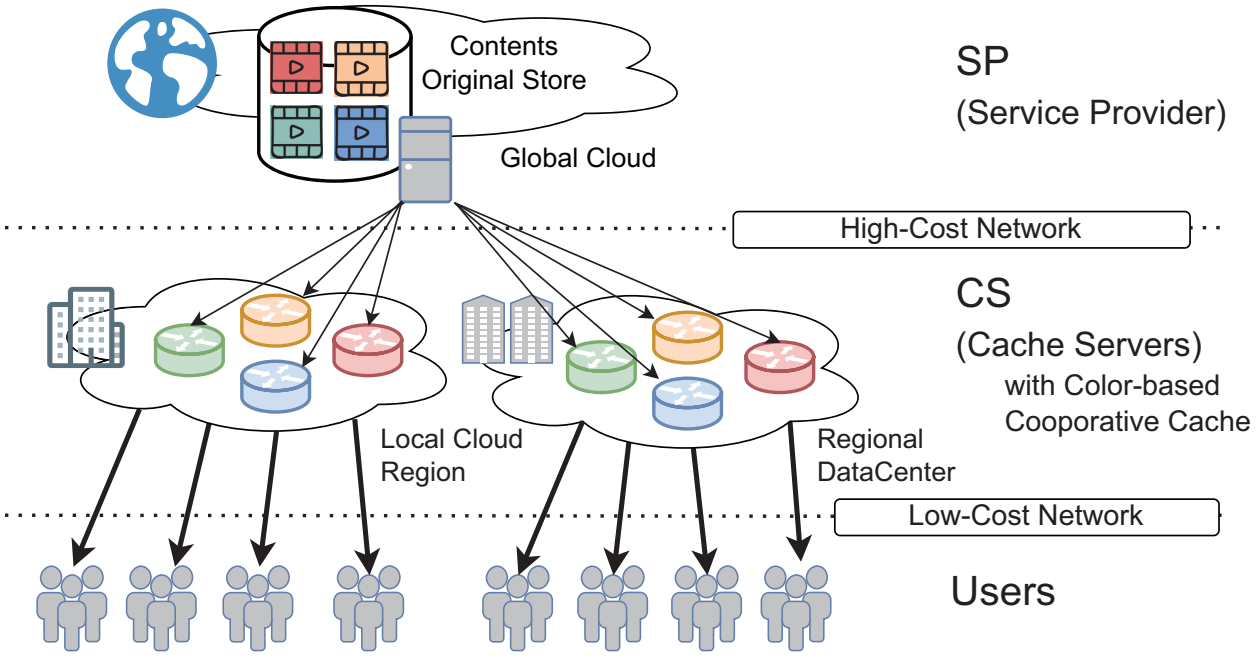


Figure 2: Hierarchical Network: An Use Case of the Color-based Cooperative Cache System with Cache-22

5.1 Color-Based Cooperative Cache System

In addition to SSL/TLS, we employ the color-based cooperative cache system [46] as the underlying cache system, which associates servers and caches through a color tag. The cache system considers both cooperative and duplicative caches. In the former system, each server caches different contents for increasing storage capacity. In the latter system, each server caches the same contents according to its popularity for increasing its hit ratio. Each cache server has a tag with a single color, and it stores contents when the server's color matches any of the contents' colors. Multiple color tags are associated to popular contents, and these contents are cached by plural servers. They propose a color tags management algorithm that updates contents' tags periodically according to their popularity ranks. As an advantage of the color-based cache system, Nakajima et al. proposed a light-weight color tags management algorithm. If we employ heuristics approach such as Genetic Algorithm (GA) for solving an optimization problem which minimizes the traffic size while satisfying several constraints such as throughput, latency, and power consumption, then they often require hours of calculation overhead. On the other hands, the overhead of the color tags management algorithm is a few minutes.

Figure 2 shows an use case of the color-based cooperative cache system which installed our Cache-22. Each entity in Figure 1 is allocated to computing node in a network hierarchy shown in Figure 2. Users can enjoy multimedia contents, such as movie, audio or rich contents for 3-D models used in XR applications distributed from storage at the global cloud. Typically, a relatively high cost is required to transfer multimedia contents directly between a cloud and users due to a volume charging pricing structure by the number of users. On the other hands, as an intranet and a local network, a traffic cost closer to users is generally low. A cache system is effective for such a hierarchical network with a cost disparity owing to a downstream suppression in a higher layer network. Introducing a color-based grouping by a content tag in a cache system makes both traffic load distributions and expanding cache capacity.

5.2 Experimental Environment

To verify the availability of Cache-22 with various ciphersuites, we implemented three types of software modules behaving as endpoints of entities, namely SP, CS, and User which has function shown in Figure 1. All modules are written in the Go language to adopt portability to various architectures and operating systems in computing nodes in the network. The implementation uses various existing libraries; however, Table 1 lists a few of the major ones needed to realize functions in the above entities.

Table 1: Libraries

	Version	Remarks
go	1.16.5	Programming Language
crypt/tls	Standard	TLS1.2/1.3 implementation
crypt/aes	Standard	AES-GCM encryption
labstack/echo	4.1.15	Web Framework
syndtr/goleveldb	1.0.0	Non-volatile key-value store

We employed post-quantum KEMs given in liboqs (version 0.6.0) [3]. Concretely, we employed Go library for TLS key exchange [4] that supports to change the TLS key exchange part to be with post-quantum security by indicating post-quantum KEMs, and employed goliboqs [5] which is a Go wrapper around liboqs. We set up an experimental environment to evaluate the performance of Cache-22. In our implementation, we fixed the maximum content length to 128 KB to realize parallel and pipeline operation for the Cache-22 system. For larger content sizes or nonaligned content, User divides the HTTPS requests by content name into multiple chunked range requests.

5.3 Evaluation: Performance Impact Caused by Employing Cache-22

As shown in Figure 3, six physical hosts were set up for the allocated role. Each host has a uniform configuration, as shown in Table 2. We supposed a cooperative cache system with four color tags by setting up four hosts to run CS module. A hosts runs User module to emulates behavior of users which issues requests to get contents. Another host is installed SP module to take a role of a contents provider.

Table 2: Host Configuration

CPU	Intel Core i7 7700	4 Core 8 Threads, 3.60GHz
Memory		32 GB
Network	(among Hosts)	1 Gbps
Storage	SSD TOSHIBA THNSNH06	60 GB
OS	Ubuntu 18.04.4	4.15.0-88-generic.x86_64

We evaluated two aspects of the performance, i.e., the throughput of User and turnaround time for which User obtains a content chunk of 128 KB. We assumed that, if a user requests a content for the first time, the content is not cached. If a cache is hit, CS returns the encrypted content; otherwise, it forwards the request to SP and receives an encrypted content to store in the cache. In our implementation, CS only forwards a request to SP because there is no duplication of the colors allocated to CS.

The experimental conditions were as follows.

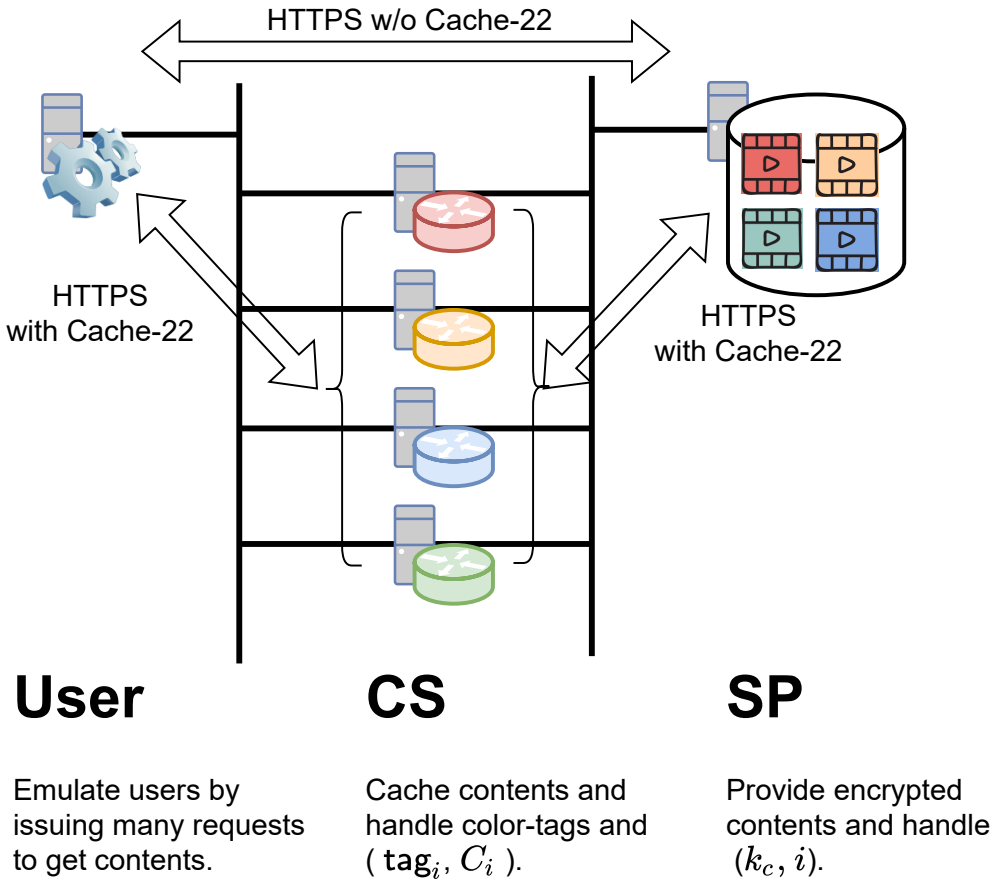
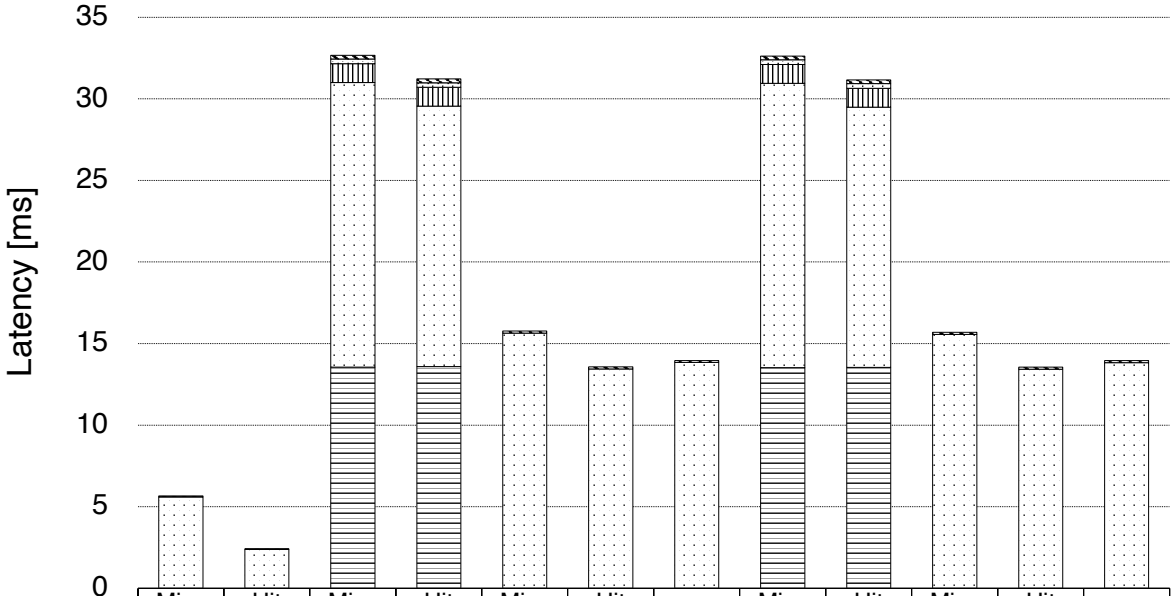


Figure 3: An Experimental Environment

- 1 GB content comprised 8,192 content chunks of 128 KB prepared by randomized data. Each tag is assigned to each content chunk.
- 120 GB content requests were generated using a gamma distribution (with a bias parameter of $k = 0.2$).
- User requested a content chunk to CS correspond to the color, that was provided by SP with tag_i.

We first evaluated five sets of performance in Figure 4 for non-post quantum ciphersuites. Communications among User, CS, and SP are encrypted by the ciphersuite indicated. We show the performance when our Cache-22 system is (resp. is not) employed in the with Cache-22 column (resp. the w/o Cache-22 column). We remark that the w/o Cache-22 case employs the MITM method, i.e., CS can observe the content and runs the color-based cache system, although all communications are encrypted. We also consider the case of HTTPS, i.e., no cache system is employed, and a user directly obtains a content from SP via HTTPS. Each row is explained as follows. Here, a user obtains (tag_i, C_i) from CS, and $k_{c,i}$ from SP.

- Decrypt: This row indicates the decryption cost of C_i using $k_{c,i}$ (this is zero in the "w/o Cache-22" and "HTTPS" columns).
- Key: This row indicates the cost of receiving $k_{c,i}$ from SP via the secure channel (this is zero in the "w/o Cache-22" and "HTTPS" columns).



	HTTP (No Security)		TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256					TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256				
	Miss	Hit	Miss with Cache-22	Hit with Cache-22	Miss w/o Cache-22 (MITM)	Hit w/o Cache-22 (MITM)	HTTPS	Miss with Cache-22	Hit with Cache-22	Miss w/o Cache-22 (MITM)	Hit w/o Cache-22 (MITM)	HTTPS
Total	5.66	2.41	32.67	31.22	15.77	13.57	13.96	32.62	31.16	15.69	13.55	13.95
HTTP Client Open/Close	0.07	0.03	0.23	0.23	0.12	0.12	0.12	0.23	0.23	0.13	0.13	0.12
Decrypt	0.00	0.00	0.28	0.28	0.00	0.00	0	0.28	0.28	0.00	0.00	0
Key	0.00	0.00	1.16	1.16	0.00	0.00	0	1.16	1.15	0.00	0.00	0
Data	5.59	2.38	17.44	15.97	15.65	13.45	13.84	17.42	15.96	15.56	13.42	13.83
Tag	0.00	0.00	13.56	13.58	0.00	0.00	0	13.53	13.54	0.00	0.00	0

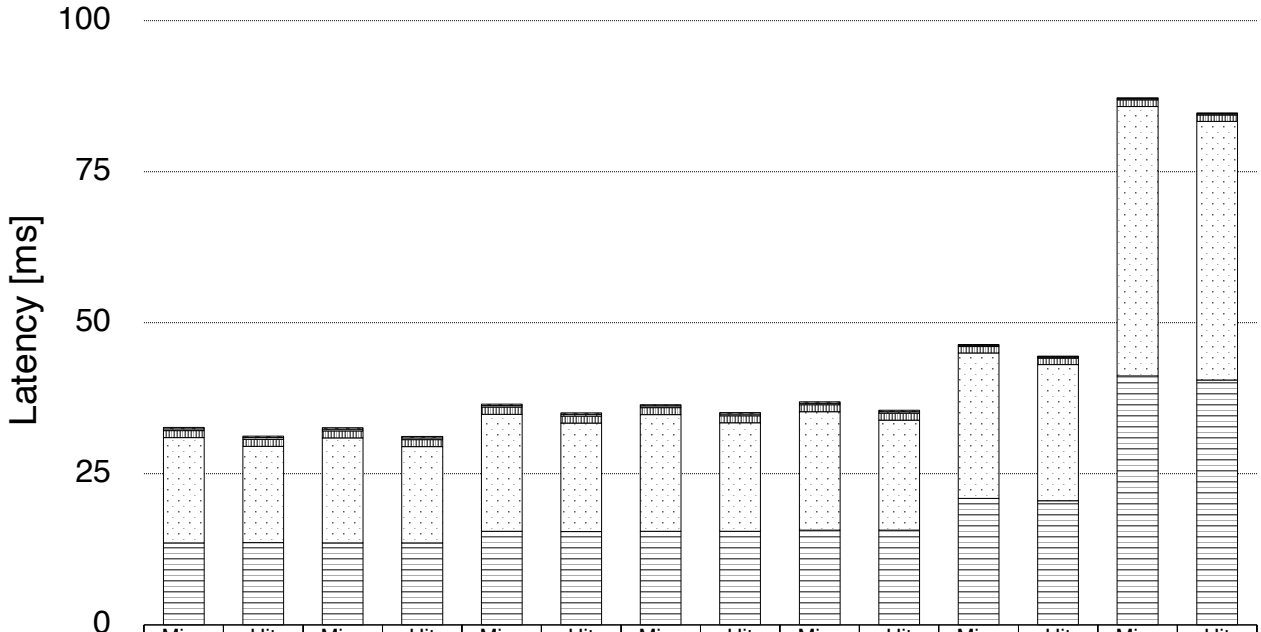
Figure 4: Evaluation of the Turnaround Time to Obtain 128 KB Chunk of Data

- Data: This row indicates the cost of receiving C_i via the secure channel in the "with Cache-22" column, and indicates the cost of receiving $\mathsf{content}_i$ via the secure channel in the "w/o Cache-22" and "HTTPS" columns.
- Tag: This row indicates the cost of receiving tag_i from SP via the secure channel (this is zero in the "w/o Cache-22" and "HTTPS" columns).

We also evaluated performance of post-quantum ciphersuites in Figure 5. The setting is the same as that of Figure 4.

5.3.1 Differences among Ciphersuites

In the case of non-post quantum ciphersuites, the difference between ciphersuites is comparatively small; however, the data transmission time using `RSA` is approximately 5%-10% longer compared to that of `ECDSA` due to the difference in the computational procedure of the cryptographic communication. In the case of post quantum ciphersuites, compared to ECDHE, two lattice-based ones, Kyber512 and LightSABER, achieve comparable efficiency, and others, especially SIKE-p434, are inefficient. Thus, from the viewpoint of efficiency, Kyber512 and LightSABER are effective.



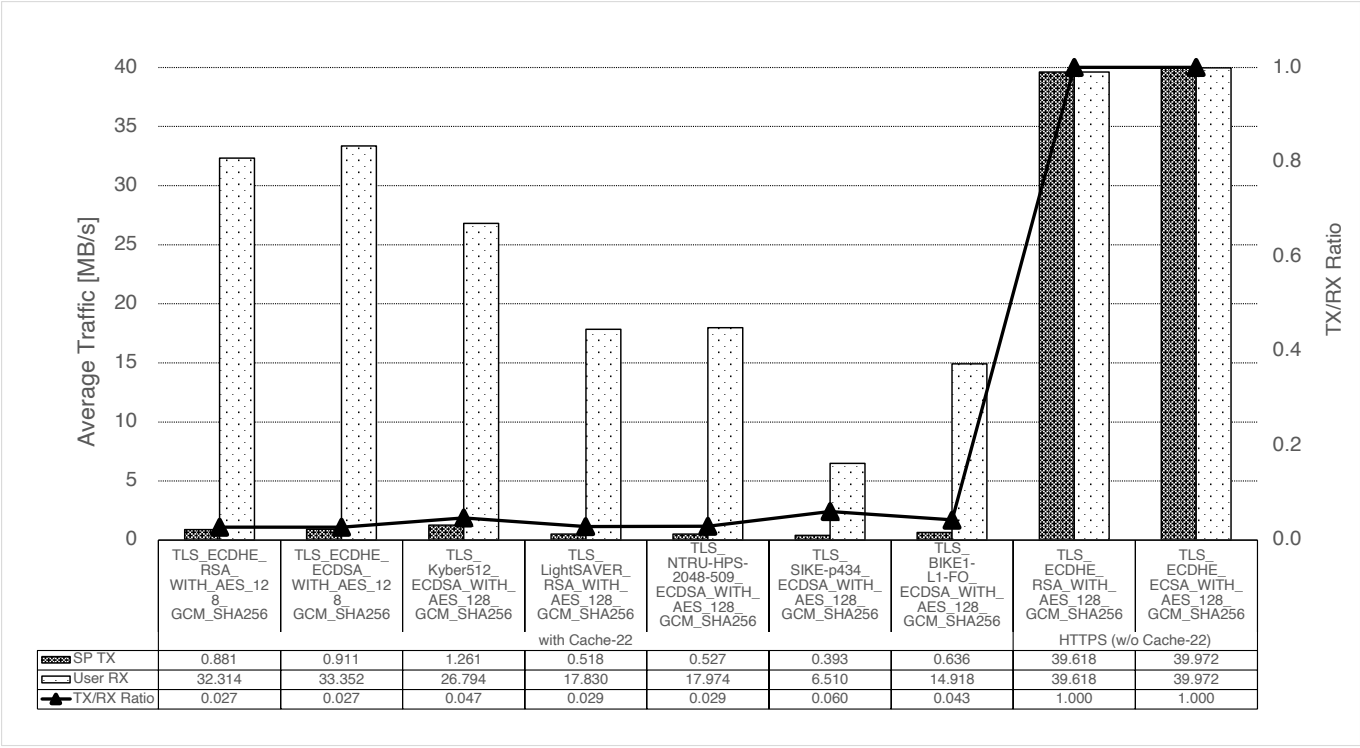
	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256		TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256		TLS_Kyber512_ECDSA_WITH_AES_128_GCM_SHA256		TLS_LightSAVER_RSA_WITH_AES_128_GCM_SHA256		TLS_NTRU-HPS-2048-509_ECDSA_WITH_AES_128_GCM_SHA256		TLS_BIKE1-L1-FO_ECDSA_WITH_AES_128_GCM_SHA256		TLS_SIKE-p434_ECDSA_WITH_AES_128_GCM_SHA256	
Total	32.67	31.22	32.62	31.16	36.53	35.07	36.44	35.12	36.92	35.52	46.41	44.45	87.23	84.74
HTTP Client Open/Close	0.23	0.23	0.23	0.23	0.23	0.23	0.23	0.23	0.23	0.23	0.16	0.15	0.15	0.15
Decrypt	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.19	0.18	0.20	0.20
Key	1.16	1.16	1.16	1.15	1.16	1.16	1.16	1.16	1.16	1.16	1.08	1.06	1.10	1.08
Data	17.44	15.97	17.42	15.96	19.40	17.95	19.29	17.95	19.57	18.17	24.08	22.53	44.61	42.79
Tag	13.56	13.58	13.53	13.54	15.46	15.45	15.48	15.50	15.68	15.68	20.90	20.53	41.17	40.52

Figure 5: Evaluation of Post-quantum Ciphersuites

However, since researches of post-quantum cryptography are still very active, we should consider cryptographic tools which are secure under several mathematical problems and its maturity. For example, in the NIST PQC document [7], it says that NTRU: *Not quite as efficient, but older* and Classic McEliece: *Oldest submission, large public keys but small ciphertexts*, that is, NTRU and Classic McEliece are good options from the viewpoint of maturity. As mentioned before, we employed BIKE and did not employ Classic McEliece as a code-based KEM owing to its large public key size. It says that BIKE: *Good performance, CCA security?, more time to be stable.* It says that SIKE: *Newer security problem, an order slower*, that is isogeny-based cryptography is a relatively new topic [21, 36] compared to other complexity problems. So, although these lack maturity, the security depends on different mathematical structures. Thus, these schemes should be considered as alternative candidates.

5.3.2 Traffic Improvement

At the first sight in Figure 4, HTTPS is the most efficient in terms of latency, and one may think that there is no merit to employ our Cache-22 system. Nevertheless, employing our Cache-22 system has a merit to drastically reduce communications between CS and SP which is effective in a hierarchical network with a cost disparity owing to a downstream suppression in a higher



	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_Kyber512_ECDSA_WITH_AES_128_GCM_SHA256	TLS_LightSAVER_RSA_WITH_AES_128_GCM_SHA256	TLS_NTRU-HPS-2048-509_ECDSA_WITH_AES_128_GCM_SHA256	TLS_SIKE-p434_ECDSA_WITH_AES_128_GCM_SHA256	TLS_BIKE1-L1-FO_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECSA_WITH_AES_128_GCM_SHA256
SP TX	0.881	0.911	1.261	0.518	0.527	0.393	0.636	39.618	39.972
User RX	32.314	33.352	26.794	17.830	17.974	6.510	14.918	39.618	39.972
TX/RX Ratio	0.027	0.027	0.047	0.029	0.029	0.060	0.043	1.000	1.000

Figure 6: Evaluation of the Traffic Efficiency

layer network (as mentioned in Figure 2). Figure 6 shows the efficiency of traffic reduction by our Cahse-22 system when we increase the number of parallel requests issued by User from one to sixteen for each experiment. Two types of bar chart indicate an average traffic transferred from SP (TX) and an average traffic received by User (RX), respectively. The ratio of TX and RX with the Cache-22 system is specified as traffic reduction rate by providing content chunks stored in CS. In the experimental conditions mentioned above, the result observed 94%-97% of cache hit ratio which means that it can save outbound data transfer prices of a high cost network in the case of Figure 2.

6 Conclusion

In this paper, we propose an encrypted cache system called Cache-22 that does not employ any additional tools beyond SSL/TLS, which yields a highly deployable system. We give not only a formal security definition but also a prototype implementation with post-quantum ciphersuites. Our implementation result shows that, compared to a direct HTTPS communication between SP and User, employing our Cache-22 system has a merit to drastically (approximately 95%) reduce communications between CS and SP which is effective in a hierarchical network with a cost disparity.

Acknowledgment: This work was supported by JSPS KAKENHI Grant Number JP21K11897.

References

[1] SIKE. Supersingular isogeny key encapsulation. `https://sike.org`.

[2] Conoha by GMO. https://www.conoha.jp/, 2019.

[3] Open Quantum Safe: Software for prototyping quantum-resistant cryptography. https://openquantumsafe.org/, June 8, 2021.

[4] Open source from Thales eSecurity: go-tls-key-exchange. https://github.com/thales-e-security/go-tls-key-exchange, May 24, 2019.

[5] Open source from Thales eSecurity: goliboqs. https://github.com/thales-e-security/goliboqs, May 24, 2019.

[6] NIST Post-Quantum Cryptography: Round 3 Submissions. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions, October 02, 2020.

[7] NIST PQC Standardization Update-Round 2 and Beyond. https://csrc.nist.gov/CSRC/media/Presentations/pqc-update-round-2-and-beyond/images-media/pqcrypto-sept2020-moody.pdf, September 23, 2020.

[8] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In *USENIX Security*, pages 327–343, 2016.

[9] D. Andreoletti, O. Ayoub, S. Giordano, G. Verticale, and M. Tornatore. Privacy-preserving caching in ISP networks. In *IEEE HPSR*, pages 1–6, 2019.

[10] D. Andreoletti, C. Rottondi, S. Giordano, G. Verticale, and M. Tornatore. An open privacy-preserving and scalable protocol for a network-neutrality compliant caching. In *IEEE ICC*, pages 1–6, 2019.

[11] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, and G. Zémor. BIKE: Bit flipping key encapsulation. https://bikesuite.org/files/BIKE.pdf, 2018.

[12] A. Araldo, G. Dán, and D. Rossi. Caching encrypted content via stochastic cache partitioning. *IEEE/ACM Trans. Netw.*, 26(1):548–561, 2018.

[13] N. Attrapadung. Dual system encryption framework in prime-order groups via computational pair encodings. In *ASIACRYPT*, pages 591–623, 2016.

[14] N. Aviram, K. Gellert, and T. Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In *EUROCRYPT*, pages 117–150, 2019.

[15] M. Bellare. New proofs for NMAC and HMAC: security without collision resistance. *J. Cryptology*, 28(4):844–878, 2015.

[16] D. J. Bernstein, T. Chou, T. Lange, I. von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, W. Wang, M. Albrecht, C. Cid, K. G. Paterson, C. J. Tjhai, and M. Tomlinson. Classic McEliece. https://classic.mceliece.org/nist.html.

[17] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144, 1998.

[18] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS - kyber: A CCA-secure module-lattice-based KEM. In *IEEE EuroS&P*, pages 353–367. IEEE, 2018.

[19] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *ACM CCS*, pages 227–238, 2014.

[20] S. Canard, A. Diop, N. Kheir, M. Paindavoine, and M. Sabt. BlindIDS: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic. In *ACM AsiaCCS*, pages 561–574, 2017.

[21] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. CSIDH: an efficient post-quantum commutative group action. In *ASIACRYPT*, pages 395–427, 2018.

[22] C. Chen, O. Danba, J. Hoffstein, A. Hulsing, J. Rijneveld, J. M. Schanck, P. Schwabe, W. Whyte, Z. Zhang, T. Saito, T. Yamakawa, and K. Xagawa. NTRU. `https://ntru.org/`.

[23] K. Cohn-Gordon, C. J. F. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. In *IEEE EuroS&P*, pages 451–466. IEEE, 2017.

[24] L. Constantin. HTTPS scanning in Kaspersky antivirus exposed users to MITM attacks. `https://www.pcworld.com/article/3154608/security/`, 2017.

[25] S. Cui, M. R. Asghar, and G. Russello. Multi-CDN: Towards privacy in content delivery networks. *IEEE Transactions on Dependable and Secure Computing*, 17(5):984–999, 2020.

[26] W. Cui, T. Chen, C. Fields, J. Chen, A. Sierra, and E. Chan-Tin. Revisiting assumptions for website fingerprinting attacks. In *ACMAsiaCCS*, pages 328–339, 2019.

[27] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren. SABER. `https://www.esat.kuleuven.be/cosic/pqcrypto/saber/`.

[28] N. Desmoulins, P. Fouque, C. Onete, and O. Sanders. Pattern matching on encrypted streams. In *ASIACRYPT*, pages 121–148, 2018.

[29] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *ICDCS*, pages 617–624, 2002.

[30] K. Emura, S. Moriai, T. Nakajima, and M. Yoshimi. Cache-22: A highly deployable encrypted cache system. In *ISITA*, pages 465–469. IEEE, 2020.

[31] G. A. Eriksson, J. Mattsson, N. Mitra, and Z. Sarker. Blind cache: a solution to content delivery challenges in an all-encrypted web. ericsson white paper, 2016.

[32] K. Hashimoto, S. Katsumata, K. Kwiatkowski, and T. Prest. An efficient and generic construction for signal's handshake (X3DH): Post-quantum, state leakage secure, and deniable. In *Public-Key Cryptography*, pages 3–33, 2021.

[33] A. Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*, pages 171–178, 2002.

[34] L. Huang, A. Rice, E. Ellingsen, and C. Jackson. Analyzing forged SSL certificates in the wild. In *2014 IEEE S&P*, pages 83–97, 2014.

[35] T. Iwata and Y. Seurin. Reconsidering the security bound of AES-GCM-SIV. *IACR Trans. Symmetric Cryptol.*, 2017(4):240–267, 2017.

[36] D. Jao and L. D. Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography*, pages 19–34, 2011.

[37] J. Jarmoc. SSL/TLS interception proxies and transitive trust. In *Blackhat Europe*, 2012.

[38] C. D. Jensen. CryptoCache: a secure sharable file cache for roaming users. In *ACM SIGOPS*, pages 73–78, 2000.

[39] S. Lai, X. Yuan, S. Sun, J. K. Liu, R. Steinfeld, A. Sakzad, and D. Liu. Towards practical encrypted network traffic pattern matching for secure middleboxes. *CoRR*, abs/2001.01848, 2020.

[40] J. Leguay, G. S. Paschos, E. A. Quaglia, and B. Smyth. CryptoCache: Network caching with confidentiality. In *IEEE ICC*, pages 1–6, 2017.

[41] G. Leurent and T. Peyrin. From collisions to chosen-prefix collisions application to full SHA-1. In *EUROCRYPT*, pages 527–555, 2019.

[42] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. Meltdown: Reading kernel memory from user space. In *USENIX Security*, pages 973–990, 2018.

[43] S. Loreto, J. Mattsson, R. Skog, H. Spaak, G. Gus, D. Druta, and M. Hafeez. Explicit trusted proxy in HTTP/2.0. In *IETF Standards-Track Internet-Draft*, 2014.

[44] D. McGrew, D. Wing, Y. Nir, and P. Gladstone. TLS proxy server extension. In *IETF Standards-Track Internet-Draft*, 2012.

[45] C. Moldovan, F. Metzger, S. Surminski, T. Hoßfeld, and V. Burger. Viability of Wi-Fi caches in an era of HTTPS prevalence. In *IEEE ICC*, pages 1370–1375, 2017.

[46] T. Nakajima, M. Yoshimi, C. Wu, and T. Yoshinaga. Color-based cooperative cache and its routing scheme for telco-CDNs. *IEICE Transactions*, 100-D(12):2847–2856, 2017.

[47] NSS Labs. NSS Labs Predicts 75% of Web Traffic Will Be Encrypted by 2019. `https://www.nsslabs.com/`, 2016.

[48] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.

[49] R. Peon. Explicit proxies for HTTP/2.0. In *IETF Standards-Track Internet-Draft*, 2012.

[50] I. M. Platform. Storing sensitive data in encrypted cache. `https://mobilefirstplatform.ibmcloud.com/`, 2016.

[51] F. Y. Rashid. Encryption, Privacy in the Internet Trends Report (June 12, 2019). `https://duo.com/decipher/encryption-privacy-in-the-internet-trends-report`, 2019.

[52] J. Reschke and S. Loreto. 'Out-Of-Band' content coding for HTTP. In *IETF Standards-Track Internet-Draft*, 2017.

[53] E. Rescorla. The transport layer security (tls) protocol version 1.3. rfc 8446. `https://rfc-editor.org/rfc/rfc8446.txt`, 2018.

[54] P. Schwabe, D. Stebila, and T. Wiggers. Post-quantum TLS without handshake signatures. In *ACM CCS*, pages 1461–1480. ACM, 2020.

[55] S. Sevilla, J. J. Garcia-Luna-Aceves, and H. R. Sadjadpour. GroupSec: A new security model for the web. In *IEEE ICC*, pages 1–6, 2017.

[56] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. BlindBox: Deep packet inspection over encrypted traffic. In *ACM SIGCOMM*, pages 213–226, 2015.

[57] Y. Shin, D. Koo, and J. Hur. A survey of secure data deduplication schemes for cloud storage systems. *ACM Comput. Surv.*, 49(4):74:1–74:38, 2017.

[58] E. Stefanov, M. van Dijk, E. Shi, T. H. Chan, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious RAM protocol. *J. ACM*, 65(4):18:1–18:26, 2018.

[59] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. The first collision for full SHA-1. In *CRYPTO*, pages 570–596, 2017.

[60] Synopsys Inc. The Heartbleed Bug. `http://heartbleed.com/`, 2014.

[61] M. Thomson, G. Eriksson, and C. Holmberg. An architecture for secure content delegation using HTTP. In *IETF Standards-Track Internet-Draft*, 2016.

[62] M. Thomson, G. Eriksson, and C. Holmberg. Caching secure HTTP content using blind caches. In *IETF Standards-Track Internet-Draft*, 2016.

[63] A. Tutoveanu. Active implementation of End-to-End post-quantum encryption. *IACR Cryptology ePrint Archive*, 2021:356, 2021.

[64] M. Usman, M. R. Asghar, I. S. Ansari, F. Granelli, Q. H. Abbasi, and K. A. Qaraqe. A marketplace for efficient and secure caching for IoT applications in 5G networks. In *2018 IEEE WCNC*, pages 1–6, 2018.

[65] Varnish Software. Introducing varnish total encryption. `https://www.varnish-software.com/varnish-total-encryption/`, 2018.

[66] X. Wang and H. Yu. How to break MD5 and other hash functions. In *EUROCRYPT*, pages 19–35, 2005.

[67] C. A. Wood and E. Uzun. Flexible end-to-end content security in CCN. In *IEEE CCNC*, pages 858–865, 2014.

[68] H. Xiong, X. Zhang, D. Yao, X. Wu, and Y. Wen. Towards end-to-end secure content storage and delivery with public cloud. In *ACM CODASPY*, pages 257–266, 2012.