CryptoCache: Network Caching with Confidentiality

Jérémie Leguay*, Georgios S. Paschos*, Elizabeth A. Quaglia†, Ben Smyth*
*Mathematical and Algorithmic Sciences Lab, France Research Center, Huawei Technologies Co. Ltd.
†Royal Holloway, University of London

***Abstract*—End-to-end encryption seemingly signifies the death of caching, because current methods ensure that no two sessions are alike. In this paper, we show that servers can reuse encrypted content between sessions, thereby rejuvenating caching. The main idea of our technique is to allow interim nodes to cache content based on pseudo-identifiers instead of real file identities. This enables caching of reusable pseudo-identifiers, whilst maintaining content confidentiality, i.e., ensuring that only the client and the server know the actual identity of the requested file. Furthermore, we provide an extension that prevents client linkability, i.e., ensuring it is impossible to tell if two clients are viewing the same content. Finally, we formally analyse the balance between security and the hit probability performance of the cache.**

***Index Terms*—Caching, Security, All-Encrypted Web, TLS.**

I. Introduction

Network caching is the act of intelligently replicating reusable content inside the network in order to improve latency and reduce network bandwidth usage. In the last decade, Content Delivery Network (CDN) providers have played a significant role in the proliferation of the Internet by replicating origin servers' content at *edge caches*, thereby reducing congestion. The leading CDN provider, Akamai, has deployed over 170,000 edge caches in more than 1,300 networks in 102 countries [1]. Such caches are typically deployed in wired networks, and deployment in wireless networks can further improve latency and reduce bandwidth [2]–[6]. In the next five years, network traffic will be dominated by video content, which is estimated to account for more than 60% of all traffic [7]. And caching will be necessary to ensure sustainability of networks. Indeed, estimates suggest CDNs will deliver 72% of video traffic by 2019 [7].

Amidst growing security concerns [8], we are currently observing a trend towards end-to-end encryption. Indeed, content originating from web giants – including Facebook, Google, Netflix, and Yahoo – is encrypted by default.[1] Indeed, Cisco predict hyper-growth in encrypted network traffic [7]. And, more concretely, Sandvine predict that 66% of North American Internet traffic will be encrypted by the end of 2016 [8]. End-to-end encryption (e.g., HTTPS/TLS) provides security, but encrypted traffic cannot be reused by the network, because encryption ensures each session is distinct. And, more generally, any operation that requires observation of (unencrypted) content is precluded. (Such operations include collecting statistics about popularity [9], for instance.)

Content and CDN providers bypass the problem of encrypted traffic by defining edge caches as end points [10]. This requires content providers to share content and cryptographic keys with edge caches, thereby allowing caches to establish end-to-end encryption with users and distribute content over encrypted connections, on behalf of content providers. This inevitably weakens security guarantees. In particular, no security is offered against a compromised edge cache. Thus, edge caches are assumed to be trusted, and untrusted edge caches cannot perform such a function (Fig. 1). Hence, the problem is assumed away for trusted edge caches.

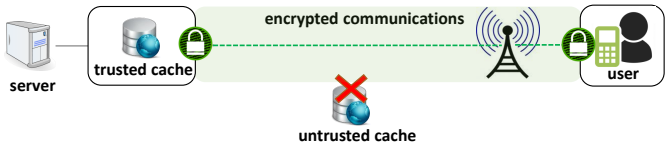

Fig. 1: End-to-end encryption precludes untrusted caches, and limits caching to trusted entities. This paper proposes a new security protocol to enable caching encrypted content and encourage the operation of untrusted caches.

Contribution. We propose CryptoCache, a security protocol that enables *caching of encrypted content*, without trusting the cache. Our protocol instructs content providers to associate content with pseudo-identifiers and to symmetrically encrypt content. Pseudo-identifiers and symmetrically encrypted content remain constant across client requests, enabling reuse. Nevertheless, the use of symmetric encryption ensures confidentiality. Since confidentiality is the main security concern in content delivery, CryptoCache combines the seemingly contradictory benefits of both security and network efficiency. Finally, we formally analyse the balance between security and caching performance.

II. CryptoCache

We consider scenarios in which a client requests content from a server and edge caches are used to improve efficiency, whilst ensuring the following property:

> *Confidentiality.* A client's request and a server's response are only revealed to the client and the server.

Confidentiality should hold even when edge caches are operated by an adversary.

A. Protocol description

We propose a security protocol to enable caching of encrypted content that satisfies confidentiality. More specifically, the protocol requires the origin server to associate each item of content file with an identifier id, a pseudo-identifier pid, and

1. https://www.eff.org/encrypt-the-web-report.

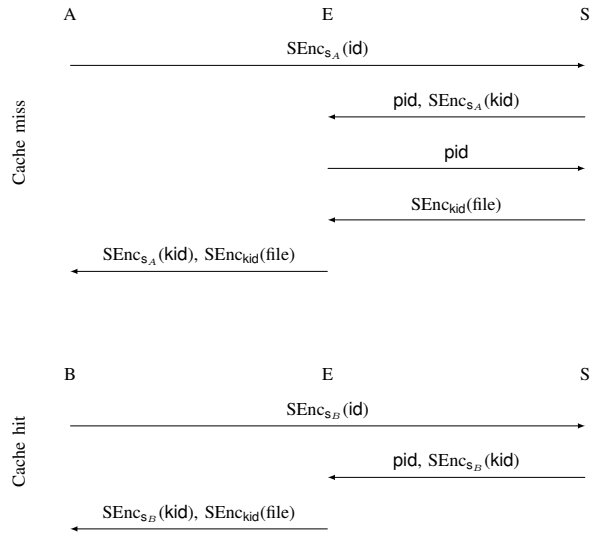

Fig. 2: CryptoCache: A protocol for caching encrypted traffic.

an encryption key kid. And edge caches to associate pseudo-identifiers with encrypted content, e.g., an edge cache might associate pid with $\mathsf{SEnc}_{\mathsf{kid}}(\mathsf{file})$.[2] The protocol is as follows:

- **Setup.** The protocol assumes a client establishes a secret session key s with a server prior to each run of the protocol. (Such keys can be established using a TLS handshake, for instance.)
- **Request.** To request content file, client C encrypts the content's identifier id with secret key s and sends the resulting ciphertext $\mathsf{SEnc}_{\mathsf{s}}(\mathsf{id})$ to server S, i.e.,
  - $C \longrightarrow S$: $\mathsf{SEnc}_{\mathsf{s}}(\mathsf{id})$
- **Response.** To respond to a request, server S decrypts ciphertext $\mathsf{SEnc}_{\mathsf{s}}(\mathsf{id})$ using session key s to recover identifier id, retrieves the corresponding pseudo-identifier pid and corresponding encryption key kid, encrypts kid with s, and sends the resulting ciphertext $\mathsf{SEnc}_{\mathsf{s}}(\mathsf{kid})$ coupled with pid to edge cache E. Namely,
  - $S \longrightarrow E$: $\mathsf{pid}, \mathsf{SEnc}_{\mathsf{s}}(\mathsf{kid})$

  If edge cache E does not hold an association between pseudo-identifier pid and some encrypted content (i.e., a cache miss occurs), then E sends pid to server S, the server responds with ciphertext $\mathsf{SEnc}_{\mathsf{kid}}(\mathsf{file})$, and E associates the pseudo-identifier and ciphertext, where pid is associated with content file and encryption key kid. That is,
  - If cache miss, then
    - $E \longrightarrow S$: pid
    - $S \longrightarrow E$: $\mathsf{SEnc}_{\mathsf{kid}}(\mathsf{file})$

  Once edge cache E has an association between pseudo-identifier pid and encrypted content $\mathsf{SEnc}_{\mathsf{kid}}(\mathsf{file})$, the edge cache sends the encrypted content along with the encrypted encryption key to client C. Namely,
  - $E \longrightarrow C$: $\mathsf{SEnc}_{\mathsf{s}}(\mathsf{kid}), \mathsf{SEnc}_{\mathsf{kid}}(\mathsf{file})$

  Upon receipt, client C decrypts ciphertext $\mathsf{SEnc}_{\mathsf{s}}(\mathsf{kid})$ using session key s to recover encryption key kid and uses kid to decrypt ciphertext $\mathsf{SEnc}_{\mathsf{kid}}(\mathsf{file})$ to recover content file, thereby concluding a run of the protocol.

Fig. 2 depicts an instance of CryptoCache in which: client A and server S share a secret key s_A, A requests id; a cache miss occurs at edge cache E, thus, the edge cache requests the encrypted content from server S and forwards the encrypted content to A; subsequently, client B and server S share a secret key s_B, B requests the same content; and edge cache E responds to B directly. Thereby demonstrating that the protocol enables encrypted content to be cached.

B. Informal security analysis

We conduct an informal security analysis of CryptoCache instantiated with a secure encryption scheme. Recall that confidentiality demands "a client's request and a server's response are only revealed to the client and the server." Our protocol transmits $\mathsf{SEnc}_{\mathsf{s}}(\mathsf{id})$, i.e., the client's request encrypted by the secret session key s established between the client and server. Thus, the security of the encryption scheme underlying our protocol ensures that the requested content identifier is only revealed to the client and the server. Our protocol also transmits $\mathsf{SEnc}_{\mathsf{s}}(\mathsf{kid})$ and $\mathsf{SEnc}_{\mathsf{kid}}(\mathsf{file})$, i.e., the encryption key associated with id encrypted by secret session key s and the content encrypted with kid. Since s is the secret session key in a protocol run, kid is only known to the client and the server. And since the server's response is encrypted by kid, the encrypted content file is only revealed to the client and server.

C. Possible implementation over HTTP

CryptoCache can be implemented over HTTP. In this setting, the client establishes a connection with the content server. And, to discover the cache, the server might provide the client with the URL of an external resource from which the encrypted content can be downloaded. The server may use application messages or the "out-of-band" content encoding HTTP option [12] to provide the client with all the necessary information: URL of the cache, pseudo-identifier of the content, and any other details. The external resource can either be a particular cache if it is known to the origin server or the one of a complete CDN. In the former case, requests are redirected by the CDN to reach the closest cache using traditional DNS mechanisms. As a consequence, CryptoCache does not require any modifications of web standards and can be implemented as an application library inside origin servers and caches.

III. CryptoCache Extension for Unlinkability

CryptoCache (§II) enables caching of encrypted content in an efficient manner, whilst ensuring confidentiality. However, the protocol permits linkability between clients requesting the same content, because the corresponding pseudo-identifier and encrypted content remain constant across requests. A higher degree of security can be obtained by satisfying the following:

2. We denote the symmetric encryption of m with key k as $\mathsf{SEnc}_{\mathsf{k}}(\mathsf{m})$, and refer the reader to Katz & Lindell [11, §3] for a formal definition.

Unlinkability. Requests for the same content cannot be detected as such, except by the edge cache.

Unlinkability permits edge caches to detect when the same content is requested by one or more clients. This exception is necessary for edge caches to perform their task.

A. Protocol description

We now adapt our protocol to satisfy confidentiality *and* unlinkability. As per the original description, we require servers to associate each piece of content with a identifier, a pseudo-identifier, and an encryption key. And edge caches associate pseudo-identifiers with encrypted content. The protocol proceeds as follows:

- **Setup.** As per Section II.
- **Request.** As per Section II.
- **Response.** To respond to a request, server S decrypts ciphertext $\mathsf{SEnc_s(id)}$ using session key s to recover identifier id, retrieves the corresponding pseudo-identifier pid and encryption key kid, generates a symmetric key t, symmetrically encrypts the concatenation of kid and t with s, asymmetrically encrypts the concatenation of pid and t with the edge cache's public key pk_E, and sends the resulting ciphertexts to the edge cache. Namely,[3]
  - $S \longrightarrow E$: $\mathsf{AEnc_{pk_E}(pid||t), SEnc_s(kid||t)}$

  Edge cache E decrypts ciphertext $\mathsf{AEnc_{pk_E}(pid||t)}$ using its private key sk_E to recover pid and symmetric key t. If a cache miss occurs, then E generates a symmetric key t', encrypts the concatenation of pid and t' with the server's public key pk_S, and sends the resulting ciphertext to server S, which the server decrypts with private key sk_S to recover pid and t', and responds with ciphertext $\mathsf{SEnc_{t'}(SEnc_{kid}(file))}$, finally, edge cache E decrypts that ciphertext with t' and associates the pseudo-identifier with the recovered encrypted content $\mathsf{SEnc_{kid}(file)}$, where pid is associated with content file and encryption key kid. That is,
  - If cache miss, then
    - $E \longrightarrow S$: $\mathsf{AEnc_{pk_S}(pid||t')}$
    - $S \longrightarrow E$: $\mathsf{SEnc_{t'}(SEnc_{kid}(file))}$

  Once edge cache E has an association between pseudo-identifier pid and encrypted content $\mathsf{SEnc_{kid}(file)}$, the edge cache symmetrically encrypts ciphertext $\mathsf{SEnc_{kid}(file)}$ with symmetric key t, and sends the resulting ciphertext coupled with ciphertext $\mathsf{SEnc_s(kid||t)}$ to client C. Namely,

  $E \longrightarrow C$: $\mathsf{SEnc_s(kid||t), SEnc_t(SEnc_{kid}(file))}$.

  Upon receipt, client C decrypts ciphertext $\mathsf{SEnc_s(kid||t)}$ using session key s to recover encryption key kid and symmetric key t, uses t to decrypt ciphertext $\mathsf{SEnc_t(SEnc_{kid}(file))}$ to recover ciphertext $\mathsf{SEnc_{kid}(file)}$, which it decrypts using kid to recover content file, thereby concluding a run of the protocol.

Fig. 3 depicts an instance of the protocol in which: client A requests id; a cache miss occurs at edge cache E, thus, the edge cache requests the encrypted content from server S and forwards the encrypted content to A; subsequently, client B requests the same request; and edge cache E responds to B directly. Thereby demonstrating that the protocol enables encrypted content to be cached.

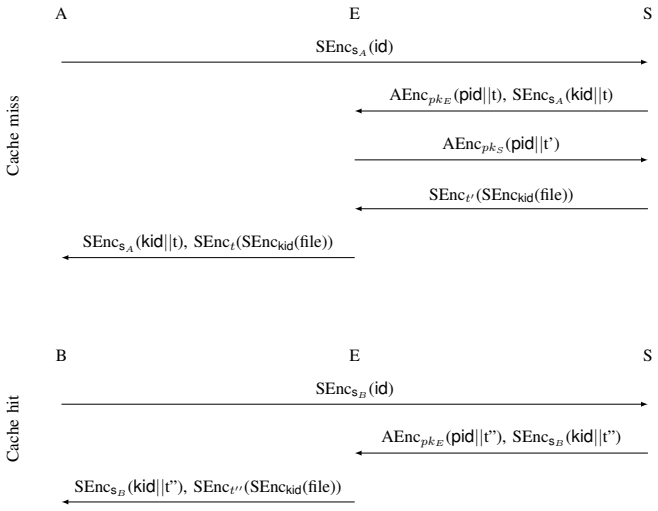

Fig. 3: Extension of CryptoCache to prevent *linkability*.

The number of messages exchanged is the same as for the first version of CryptoCache and it achieves identical caching performance. However, extra encryptions are required, which increases computational complexity.

B. Informal security analysis

When instantiated with a secure symmetric encryption scheme and a secure asymmetric encryption scheme, our extension to CryptoCache satisfies confidentiality due to reasons similar to those presented in Section II-B. Furthermore, unlinkability is satisfied, because pseudo-identifiers are asymmetrically encrypted, thus, cannot be linked. Moreover, encrypted content is distinct between sessions due to nested encryption. In particular, requests by edge caches for encrypted content cannot be linked, nor can requests by clients.

Our protocol could be easily extended to satisfy *authentication* and *integrity*. Authentication can be achieved by the user authenticating the server during the establishment of the secret session key in the setup phase. Thus, authentication can be provided in a standard way, e.g., public-key certificates. Integrity can also be achieved in a standard way, e.g., using message authentication codes.

IV. Balancing caching performance and security

To perform their task, edge caches necessarily link pseudo-identifiers with encrypted content, between client requests. Consequently, it is possible for an edge cache to exhaustively search a server's identifier space to map identifiers to pseudo-identifiers and encrypted content. (The edge cache might conduct an exhaustive search by sending messages directly to

3. We write $m||n$ for the concatenation of messages m and n. And denote the assymmetric encryption of m with public key pk as $\mathsf{AEnc_{pk}(m)}$.

time slot	1	2	3	4	5	6	7
search for file n	0	0	1	0	0	0	0
update pseudo-identifier of file n	0	0	0	0	1	0	1
pid of file n	a	a	a	a	b	b	c
kid of file n	$\hat{a}$	$\hat{a}$	$\hat{a}$	$\hat{a}$	$\hat{b}$	$\hat{b}$	$\hat{c}$
output of process $X(n,t)$	0	0	1	1	0	0	0

TABLE I: Exemplar evolution of process $X(n,t)$.

the server, or by colluding with one or more users to do so.) Thus, edge caches might learn which content clients request. Servers can invalidate such mappings, simply by updating pseudo-identifiers and encryption keys. For instance, servers might update pseudo-identifiers and keys after every request. However, updates have repercussions on caching performance: if a file exists in the cache with an obsolete pseudo-identifier, a request for this file will be a miss. Hence, frequent updates decrease hit rate. In this section, we balance this trade-off by answering the following question: *how often should updates occur whilst keeping an acceptable hit probability performance?* We use a Stackelberg security game to answer this question. More precisely, given a cache hit probability target, we establish an update strategy which minimizes the probability of a successful search, i.e., a search that successfully maps content identifiers to pseudo-identifiers and encrypted content.

A. Search and update strategies

We consider slotted time using slots $t = 1, 2, \ldots$ corresponding to time intervals $[0, T], [T, 2T], \ldots$, where T is the slot size. We suppose requests for N files are made with a discretized *Independent Reference Model* (IRM) [13]. At each slot, exactly one request for a file is made.[4] The request is for file n with probability p_n.

Consider a stochastic process X that inputs a file n and a time slot t, and outputs 1 if the pseudo-identifier associated with n has been revealed and outputs 0 otherwise. We remark that after updating a pseudo-identifier, X will output zero. We exemplify the evolution of the stochastic process in Table I.

We define the *successful search* event A as follows: we let time grow $t \to \infty$ and consider the confidentiality of an arbitrary request Y, where $\mathbb{P}(Y = n) = p_n$. The search is successful if Y's pseudo-identifier is revealed, i.e., if $X(Y, \infty) = 1$. Formally, the *probability of a successful search* is defined as follows:

$$\begin{aligned}\mathbb{P}_A &= \limsup_{t\to\infty} \mathbb{E}_Y \left[X(Y,t) = 1\right] \\ &= \limsup_{t\to\infty} \sum_n p_n \mathbb{P}(X(n,t) = 1)\,.\end{aligned}$$

Henceforth, we restrict ourselves to randomized update/search strategies. Let t be a slot, let $d_n \in [0,1]$ be the probability to update the pseudo-identifier of file n, and let $a_n \in [0,1]$ be the probability to search for file n. Moreover, let $\boldsymbol{d}, \boldsymbol{a}$ be the corresponding vectors of probabilities. In this context, process $X(n,t)$ is a binary Markov chain with state transition probabilities $p_{01} = a_n(1-d_n)$ and $p_{10} = d_n$.

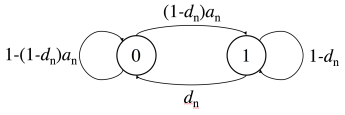

Fig. 4: Binary Markov chain modelling the evolution of the stochastic process $X(n,t)$.

(Fig. 4), and we compute the stationary probability as follows:

$$\mathbb{P}(X(n,\infty) = 1) = \begin{cases} 0 & \text{if } a_n = 0 \text{ or } d_n = 1 \\ \frac{a_n}{a_n + \frac{d_n}{1-d_n}} & \text{otherwise.} \end{cases}$$

Hence, $\mathbb{P}_A$ reduces as follows:

$$\mathbb{P}_A(\boldsymbol{d}, \boldsymbol{a}) = \sum_{\substack{n=1..N \\ n:d_n<1}} p_n \frac{a_n}{a_n + \frac{d_n}{1-d_n}}.$$

The update strategy $\boldsymbol{d}$ should be chosen to minimize $\mathbb{P}_A$, while the search strategy $\boldsymbol{a}$ to maximize it. Hence, performance is ultimately characterized by a two-player zero-sum game.

Consider the following realistic setting. The search strategy's impact on $X(n,t)$ is naturally not observable by the update strategy. On the contrary, it is reasonable to assume that the update strategy is fixed and advertized (e.g. standardized), and hence known to the entity that decides $\boldsymbol{a}$ (here assumed to be the edge cache). Indeed, the server should be *a priori* designed to withstand the worst-case search strategy. In this context, characterizing the performance of $\mathbb{P}_A(\boldsymbol{d}, \boldsymbol{a})$ falls into the classical framework of Stackelberg security games [14]. Such games can be solved as follows. The optimal update strategy $\boldsymbol{d}^*$ is given by

$$\boldsymbol{d}^* \in \arg\min_{\boldsymbol{d}} \mathbb{P}_A(\boldsymbol{d}, BR_a(\boldsymbol{d})),$$

where $BR_a(\boldsymbol{d})$ is the best response search strategy for $\boldsymbol{d}$. Thus, the optimal search strategy is $BR_a(\boldsymbol{d}^*)$. The pair of strategies $(\boldsymbol{d}^*, BR_a(\boldsymbol{d}^*))$ is called a *Strong Stackelberg Equilibrium* (SSE), and is known to coincide with the following optimization [14], [15]:

$$\min_{\substack{0\le d_n\le 1 \\ g_1(\boldsymbol{d})\le 0}} \max_{\substack{0\le a_n\le 1 \\ g_2(\boldsymbol{a})\le 0}} \sum_{\substack{n=1..N \\ n:d_n<1}} p_n \frac{a_n}{a_n + \frac{d_n}{1-d_n}}. \tag{1}$$

We proceed to clarify the constraints g_1, g_2 in our application.

The hit probability performance is directly affected by the updates of pseudo-identifiers. Let $h_n(\boldsymbol{d})$ be the hit probability of file n with respect to update strategy $\boldsymbol{d}$. Probability $h_n(\boldsymbol{d})$ depends on the used caching policy, here we consider the *cache the most popular files* policy.[5] We have

$$h_n(\boldsymbol{d}) = \begin{cases} (1-d_n), & n = 1, \ldots, C \\ 0, & n = C+1, \ldots, N. \end{cases}$$

4. If we generate requests for files with probability $\lambda \in [0,1]$ and let the slot size T be arbitrarily small, our model approaches the classical continuous time IRM with rate λ/T. Here we drop λ since it is a mere scaling factor that does not affect our analysis.

5. The computation of $h_n(\boldsymbol{d})$ for other policies (e.g., LRU and LFU) is a possible direction for future work.

Note that term $(1-d_n)$ equals the probability that a request for a cached file is missed due to the pseudo-identifier update. We let

$$g_1(\boldsymbol{d}) = h_{\min} - \sum_{n=1}^{C} p_n(1-d_n) = \sum_{k=1}^{C} p_k d_k - (h_C - h_{\min})$$

where $h_C = \sum_{n=1}^{C} p_n$ is the hit probability of "cache the most popular" without updates. The constraint $g_1(\boldsymbol{d}) \leq 0$ establishes that the modified hit probability due to the updates which is $\sum_{n=1}^{C} p_n(1-d_n)$ is more than $h_{\min}$, the desired hit probability. Note that if we do not require achieving a positive hit probability, we may set $h_{\min} = 0$ and then the optimal update strategy is to update all pids continuously: $d_n = 1$, for all n.

Typically, there is a budget $A_{\max}$ for searches per slot. If $A_{\max} = N$, all files are requested at each slot. However, such an extreme intensity of requests can be detected by traffic analysis and then neutralized. Hence, to avoid detectability practical search strategies would limit $A_{\max}$. Thus,

$$g_2(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - A_{\max},$$

hence $g_2(\boldsymbol{a}) \leq 0$ implies the search is less than the budget $A_{\max}$. Note that if $A_{\max} = N$, the optimal search strategy is $a_n = 1$, for all n.

B. Upper bound on successful search

We derive the worst-case caching performance by solving the Stackelberg game under the assumption that the update strategy is designed for $A_{\max} = N$, i.e., the largest possible search budget. This will provide a performance bound because the update strategy could be improved if we could know a priori the actual budget of the search, which is possibly $A_{\max} < N$.

When $A_{\max} = N$, the search strategy is trivially found to be $\boldsymbol{a}^* = \mathbf{1}$, irrespective of the update strategy, hence, the optimal update $\boldsymbol{d}^*$ is presented in the following lemma.

Lemma 1 (Unlimited budget SSE): Best response update strategy $\boldsymbol{d}^* = BR_d(\mathbf{1})$ for search $\boldsymbol{a}^* = \mathbf{1}$ is

$$d_n^* = \begin{cases} 1 - \frac{h_{\min}}{h_C} & 1 \leq n \leq C \\ 1 & C+1 \leq n \leq N. \end{cases}$$

Moreover, the pair of strategies $(BR_d(\mathbf{1}), \mathbf{1})$ is a Strong Stackelberg Equilibrium when $A_{\max} = N$.

Proof of Lemma 1: With the largest budget, the search strategy is trivially derived as $a_n^* = 1$, for all n. Hence, (1) becomes

$$\min_{\substack{0 \leq d_n \leq 1 \\ \sum_{k=1}^{C} p_k d_k - \leq h_C - h_{\min}}} \sum_{\substack{n=1..N \\ n:d_n<1}} p_n \frac{1}{1 + \frac{d_n}{1-d_n}}$$

which is the best response update strategy for $\boldsymbol{a}^* = \mathbf{1}$. This is equivalent to maximizing $\sum_{\substack{n=1..N \\ n:d_n<1}} p_n d_n$, subject to the above constraints. And this problem can be solved by Karush-Kuhn-Tucker conditions to yield the result. ■

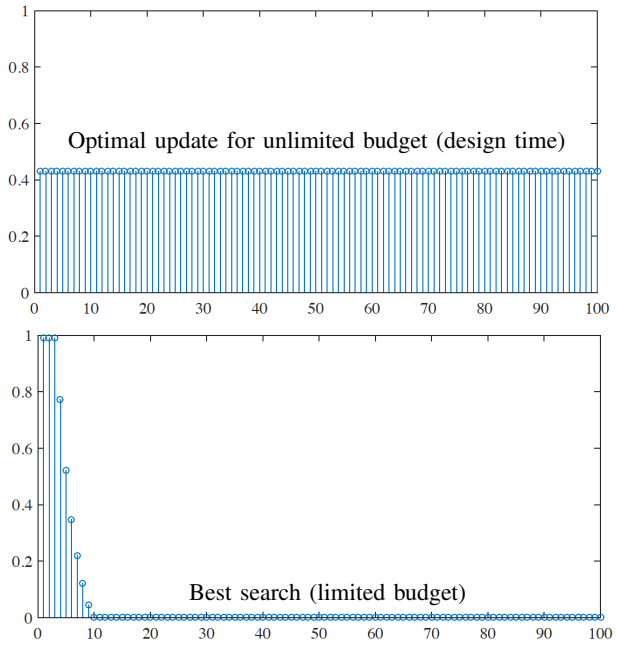

Fig. 5: Optimal update strategy $BR_d(\mathbf{1})$ for an unlimited search budget (top) and best response search strategy $BR_a(BR_d(\mathbf{1}))$ for a limited budget (bottom).

It might seem surprising that the update strategy treats all files equally (Fig. 5), and does not update popular files more frequently. This is attributed to the fact that popular files also yield big hit probabilities, hence if updated frequently, more cache misses will occur, which quickly consumes $h_{\min}$. Thus, the update strategy treats all files equally.

Having fixed the update strategy, we proceed to clarify a realistic search strategy with budget $A_{\max} < N$. This is given by $\boldsymbol{a}^* = BR_a(\boldsymbol{d}^*)$ and can be obtained using the first-order optimality condition of convex optimization, which leads to a "waterfilling" algorithm that incrementally assigns resource in steps to the file n^* with $d_{n^*} < 1$ that maximizes the partial derivative

$$\frac{\partial \frac{p_n a_n}{a_n + \beta_n(\boldsymbol{d})}}{\partial a_n} = \frac{p_n \beta_n(\boldsymbol{d})}{(a_n + \beta_n(\boldsymbol{d}))^2},$$

where $\beta_n(\boldsymbol{d}) = \frac{d_n}{1-d_n}$, and while respecting the box and simplex constraints. Fig. 5 showcases the result. The search strategy reasonably assigns more resources to popular files since in this case we have $A_{\max} < N$.

Performance is ultimately determined by successful search probability $\mathbb{P}_A$, which is a function on the popularity distribution $[p_n]$, the cache size C, the minimum hit probability requirement $h_{\min}$, and the budget $A_{\max}$. Below we fix $N = 1000$, $C = 100$ (corresponding to caching 10% of files), and popularity distribution $p_n \propto n^{-s}$, where the power law exponent $s = 0.8$, as typically measured in Internet applications with user-generated content (e.g., YouTube) [16]. The resulting $\mathbb{P}_A$ is shown in Fig. 6 for different values of $h_{\min}$, and $A_{\max}$ (given as the fraction of the catalogue N searched at each slot).

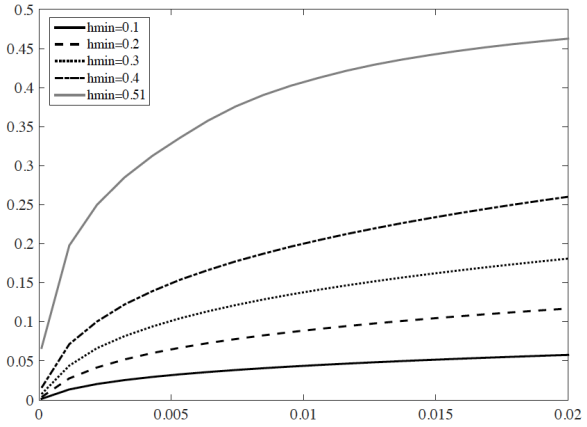

Fig. 6: Probability of successful search (y-axis) for search rate (x-axis) and hit probabilities $h_{\min} \in \{0.1, 0.2, 0.3, 0.4, 0.51\}$, with simulation parameters $s = 0.8$, $N = 1000$, and $C = 100$. We mention that the maximum hit probability in this case is $h_C = 0.52$.

For low hit probability requirements such as $h_{\min} = 0.1$, $\mathbb{P}_A$ is always smaller than 5%. Moreover, the successful search probability is small if $A_{\max} < 0.001$, that is, the search happens on the same time scale as the requests.

V. Related work

The problem of encrypted traffic can be assumed away for caches that are trusted with content providers' content and cryptographic keys (Section I). Traditional CDN solutions [10] deliver SSL content to end-users on behalf of producers. Similarly, Naylor *et al.* [17] propose multi-context TLS (mcTLS), a modified version of TLS which supports trusted edge caches. In contrast with previous work, we assume caches are untrusted. Thus, content providers' content and cryptographic keys cannot be shared with caches. Indeed, this violates our confidentiality requirement, i.e., client requests and server responses (including content) are only revealed to clients and servers. In contrast with previous work, we assume caches are untrusted. Thus, content providers' content and cryptographic keys cannot be shared with caches. Indeed, this violates our confidentiality requirement, i.e., client requests and server responses (including content) are only revealed to clients and servers.

Content producers such as Netflix share encrypted content with caches and clients request encrypted content from those caches,[6] thus a client's request is revealed to the cache, whilst the content is not. Wood & Uzun [18] use a similar technique that additionally asymmetrically encrypts the symmetric keys used to encrypt content and uses proxy re-encryption to distribute those asymmetrically encrypted keys to clients. These approaches assume caches are trusted not to reveal requests.

In parallel with our work, a technical report describing an application layer protocol for caching encrypted traffic has been released [19]. By comparison, we propose a protocol that is independent of the OSI stack. In addition, we identify a limitation on the confidentiality that can be assured when caching encrypted traffic and propose a methodology to optimise confidentiality, whereas parallel work does not.

VI. Conclusion

We have proposed CryptoCache, a security protocol to enable caching of encrypted content. It allows caches to store encrypted content with reusable pseudo-identifiers while maintaining confidentiality. The protocol provides a solution to caching in the all-encrypted web. Furthermore, we have presented an extension to prevent the linkability of user requests. Finally, we have formally analyzed the trade-off between security and the effectiveness of caching.

References

[1] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *ACM SIGCOMM CCR*, vol. 45, no. 3, 2015.

[2] G. Paschos, E. Bastug, I. Land, G. Caire, and M. Debbah, "Wireless caching: technical misconceptions and business barriers," *IEEE Communications Magazine*, vol. 54, no. 8, pp. 16–22, 2016.

[3] S. Gitzenis, G. S. Paschos, and L. Tassiulas, "Asymptotic laws for joint content replication and delivery in wireless networks," *IEEE Trans. Inf. Theory*, vol. 59, no. 5, pp. 2760–2776, 2013.

[4] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, 2014.

[5] M. Ji, G. Caire, and A. F. Molisch, "Fundamental limits of caching in wireless d2d networks," *IEEE Trans. Inf. Theory*, vol. 62, no. 2, pp. 849–869, 2016.

[6] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[7] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update 2014-2019," 2015.

[8] "Sandvine, report on global internet phenomena spotlight: Encrypted internet traffic. white paper." 2016.

[9] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the "s" in https," in *ACM CoNext*, 2014.

[10] A. van Kesteren, Secure Content Delivery Network. Akamai Whitepaper., 2014.

[11] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[12] J. Reschke and S. Loreto, "Out-Of-Band Content Coding for HTTP," Draft. Out-Of-Band Content Coding for HTTP, 2015.

[13] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," *CoRR*, vol. abs/1202.3974, 2012.

[14] D. Korzhyk, Z. Yin, C. Kiekintveld, V. Conitzer, and M. Tambe, "Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness," *J. Artif. Int. Res.*, vol. 41, no. 2, 2011.

[15] A. Ghosh and S. Boyd, "Minimax and convex-concave games," Tech. Rep., 2003.

[16] S.-E. Elayoubi and J. Roberts, "Performance and cost effectiveness of caching in mobile access networks," in *ACM-ICN*, 2015.

[17] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste, "Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS," *SIGCOMM CCR*, vol. 45, no. 4, 2015.

[18] C. A. Wood and E. Uzun, "Flexible end-to-end content security in CCN," in *IEEE CCNC*, 2014.

[19] G. 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.

6. https://pomelollc.wordpress.com/2009/04/15/on-netflixs-video-streaming-security-framework/