On the Privacy Leakage of Coded Caching

Yu Wang and Alhussein A. Abouzeid
ECSE Department
Rensselaer Polytechnic Institute
Troy, NY, USA
{wangy52, abouza}@rpi.edu

***Abstract*—Coded caching, introduced by Maddah-Ali and Niesen, jointly designs content placement and delivery policies that achieve order-optimal transmission rate. We first prove that the coded caching scheme cannot protect the anonymity of network users when an eavesdropper in the network has access to the broadcast channel. Then, a three-stage attack is proposed to estimate users file requests by building an X-D table mapping between server transmissions X and users requests D. For a coded caching network having N files and K users, there are N^K different demand vectors. We show that the approximate average time complexity of the proposed attack is only a factor of two from the total number of possible demand vectors, while the best case (for the eavesdropper) time complexity is in the order of only NK. Once the X-D table is completely deduced by the eavesdropper, the users seize from having any privacy. Numerical simulations demonstrate that the analytical results for the time complexity of the attack match well with the simulation results.**

***Index Terms*—Coded caching, anonymity, privacy.**

I. Introduction

Content-centric services, such as online movies, live TV and software applications download, take up an increasing share of the overall traffic of the internet [1]. To lower the peak load of the network, proactive caching or prefetching is one of the effective techniques that stores popular content at the user's local storage during off-peak periods in order to reduce the bandwidth demands during peak periods. The decisions regarding which files to cache and how to deliver files efficiently have been intensively studied. Schemes for different network setups have been proposed, e.g., [2]–[4]. These caching algorithms are referred to as *uncoded.* Naturally, uncoded caching schemes aim at maximizing (minimizing) the probability of *cache hit* (*cache miss*) per user to minimize the transmission rate from the server in response to misses. A significant portion of this body of work treats users as isolated individuals and analyzes the performance of each user independently.

Unlike uncoded caching, Maddah-Ali and Niesen introduced *coded caching* in [5]. In the coded caching scheme, the server uses coded transmissions on the broadcast channel to communicate with all users simultaneously, and coding is used to maximize the total caching gain. Specifically, in the placement phase, each user's cache is filled with carefully chosen subfiles. In the delivery phase, the server sends XORed subfiles according to what users have and what they want. XOR here refers to the binary exclusive or operation. It is also possible to store coded subfiles. By elaborately designing the choice of content placement in the caches during the placement phase, one XORed transmission may satisfy several users requests. Consequently, the server transmission rate over the broadcast channel is reduced compared to uncoded caching. Their work is later extended to several other important variations, such as decentralized network [4], heterogeneous cache size [6], arbitrary file popularities [7], and other generalized results [8]–[13].

However, the joint design of the cache content placement and the transmission of (sub)files on the broadcast channel causes security issues and privacy leakage. The broadcast property of the channel may leak information about files not requested by the user. Furthermore, the leaked information can be utilized by an attacker to successfully perform various privacy attacks, such as predicting file content and the user's identity. Different methods to protect privacy under the coded caching setup have been proposed in [14]–[16]. In [14], the authors encrypt each file block with a shared key. The wiretapper who has no information about the key cannot gain any information about the files transmitted. The authors of [15] manipulate phantom users to express a modified content popularity distribution. In [16], an algorithm that makes use of the idea of *secret sharing* is proposed. In their algorithm, no user can get any information about other users' file requests unless a sufficient amount of users are compromised. Finally, [17], [18] investigate achievable coded caching schemes whereby no user learns any information about any other users' demands. However, in real applications, there are cases where files in the network are public, which means content security is not the primary concern. For instance, a user sitting at home can watch any public video she likes from the Internet, and so does the eavesdropper. What the user wants is to protect is her watching history from being detected, e.g. by a neighbor in a nearby home with shared access to the wireless channel. Besides that, the questions of whether or not coded caching leaks users identity (anonymity) and file requests, and how to utilize the leakage, have not been formally studied. A related problem termed 'private information retrieval' is investigated in [19]–[22]. Under that setup, the network scenario is that one user uses multiple unicast channels to communicate with multiple databases (servers), while the coded caching network addresses a different set-up which has a single broadcast channel to connect multiple users to one server.

In this paper, we propose an attack algorithm such that an eavesdropper is able to eventually know all users' file requests by eavesdropping on the server transmission. We follow the same coded caching set-up as [5] but add one eavesdropper to the users list. Firstly, we introduce the X-D table. In the coded caching set-up, the server can be modeled as following an X-D table that generates X in response to users file requests D. Secondly, we show that once the eavesdropper constructs the X-D table, he is able to look up the X-D table to find out any other user's file request, i.e. looking up D that corresponds to server transmission X. To show the feasibility of the attack, we prove that the mapping between X and D is one-to-one. Obviously, the critical part of the attack is to construct the X-D table.

The eavesdropper's algorithm to collect privacy leakage information and to ultimately compose the X-D table is summarized as follows. The algorithm is inspired by the idea of "Differential Attack" in [23], where the eavesdropper gains some knowledge about one user's cache content by comparing two specific server transmissions. The table construction operates in three stages: 1) The eavesdropper builds a table to map coded subfiles in X to sets of file indexes; 2) The eavesdropper keeps capturing transmission sequences in X until one "anchor" sequence and all the corresponding "support" sequences are captured. (The concepts of anchor and support sequences are described in detail in the analysis sections.) By analyzing the difference in file indexes between coded files in the anchor sequence and the support sequence, the eavesdropper knows some partial information about users' cache content. At the end of this stage, the eavesdropper knows the cache contents $\mathscr{X}$ of all users; 3) By applying a formula to generate X from all possible users file requests D and $\mathscr{X}$, the eavesdropper builds the X-D table.

The contributions of this paper are thus twofold. 1) We present an attack whereby an eavesdropper makes use of the anonymity leakage of coded caching to estimate users file requests in the network. 2) We provide an analytical approximation of the attack time which fits well with the numerical simulations.

The remainder of the paper is organized as follows. The problem setup is defined in Section II. Sec. III illustrates and verifies how to use the X-D table to predict user requests. The attack algorithm to construct the X-D table is presented in Sec. IV-A. The complexity of the algorithm is analyzed in Sec. IV-C and an example of an attack is presented in Section IV-B. Section V concludes the paper and outlines avenues for future work. The Appendix contains the proof of the correctness of the algorithm and details of the complexity analysis of the algorithm.

II. Problem Formulation

In this section, we formulate the coded caching with eavesdropper problem. We describe the classic coded caching scheme, which is going to be the target of the attack, and we describe the eavesdropper attack model.

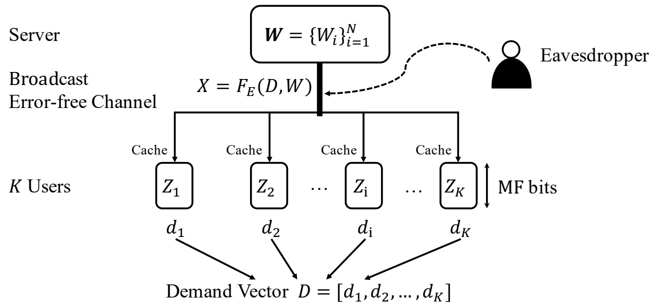
Server
$\boldsymbol{W} = \{W_i\}_{i=1}^N$
Eavesdropper
Broadcast Error-free Channel
$X = F_E(D,W)$
Cache
K Users
Z_1
Z_2
Z_i
Z_K
MF bits
d_1
d_2
d_i
d_K
Demand Vector $D = [d_1, d_2, \ldots, d_K]$

Fig. 1: One central server with N files connects K users using an error-free broadcast channel. Each user has a local cache of size MF bits. An eavesdropper would like to detect all other users demands by monitoring the server transmissions.

The coded caching network has a central server connected to $K \in Z^+$ users via an error-free broadcast channel, where Z^+ is the set of all positive integers. The server has $N \in Z^+$ files in its storage, and each of these files is of same length $F \in Z^+$ bits. A file with index n is denoted by W_n, where $n \in [N] \triangleq \{0, 1, 2, \cdots, N-1\}$. W_n is assumed to be chosen randomly from $[2^F] \triangleq [0, 1, 2, \cdots 2^F - 1]$ with equal probability. Each user has a local cache memory of size MF bits. The cache content of user i is denoted by Z_i and

$$\mathscr{X} = \{Z_0, Z_1, \cdots, Z_{K-1}\}$$

is used to represent a set containing all K users cache contents. The network structure is illustrated in Fig. 1.

Similar to the setting in [5] , the system has placement phase and delivery phase.

A. The Placement Phase

Each of the N files is split into $\binom{K}{t}$ non-overlapping subfiles. In the Placement Phase, the caches of users are initialized in such a manner that each subfile is cached by exactly t users, where $t = MK/N$. For simplicity, t here is assumed to be a positive integer. Only the server and user k know the cache content Z_k. The placement phase is assumed to have no security and privacy issues. Since the placement phase is assumed to be an off-peak phase, this could be achieved, for example, by having the server send the subfiles to users via a secure unicast channel.

B. The Delivery Phase

During the delivery phase, each user requests one file index from the server via a secure and private channel. The file index requested by user k is denoted by $d_k \in [N]$. The server collects a demand vector D from all K users noted as

$$D = [d_0, d_1, \cdots, d_{K-1}].$$

After that, the server generates coded subfile X_i for all user sets S_i, where $i \in [\binom{K}{t+1}]$, following

$$X_i = \oplus_{s \in S_i} W_{d_s, S_i \setminus \{s\}}, \tag{1}$$

where X_i is the i^{th} coded subfiles; S_i is a set containing $t+1$ user indexes; $S_i \setminus \{s\}$ is a set containing all user indexes in S_i except s; $W_{d_s, S_i \setminus \{s\}}$ represents the subfile which is in the cache memory of users whose indexes are in $S_i \setminus \{s\}$; $\oplus$ is the operator of bit-wise XOR.

The server concatenates all the coded subfiles one after another to generate the bit sequence

$$X = [X_0, X_1, \cdots, X_{\binom{K}{t+1}-1}],$$

where X is of length RF. Finally, X is sent to users through the broadcast channel.

The process of generation of X from D and $\mathscr{X}$ can be viewed as searching for X that corresponds to D in a two-column X-D table.

After receiving X from the channel, each user, say user k, makes an estimation $\hat{W}_{d_k}$ of the file W_{d_k} she requested, based on X and her own cache content Z_k.

One epoch refers to a process containing 1) users sending file requests to form the demand vector D; 2) the server sending bit sequence X in return; 3) users decoding files W_{d_k}. When one epoch is over, the server and users are ready for another epoch.

For the following analysis, we only consider the case where the number of users is no larger than the number of files, i.e. $K \leq N$. We also assume that the file popularities are the same for each user. If there are dependency between users' file requests or the file popularities are non-uniform, only the analysis related to the time complexity needs changing, but the attack algorithm proposed here still applies.

C. Eavesdropper Model

In the network, there is an eavesdropper who has access to the broadcast channel and wants to predict all other users file requests.

We have the following assumptions about the eavesdropper:

1) Assumption 1: The eavesdropper initially knows nothing about users cache contents $\mathscr{Z}$;
2) Assumption 2: The eavesdropper has error-free access to the server broadcast transmissions X;
3) Assumption 3: The eavesdropper has a sufficiently large memory and computational power.

Because the placement phase is usually conducted during the off-peak time, the server can even adopt unicast communication with encryption to send placement bits to each user. Assumption 1 is practical under such scenarios. Since the eavesdropper can perform sniffing of the broadcast channel or even (pretend to) be a legitimate user, having access to the traffic on the broadcast channel without error (Assumption 2) is also possible. In this section, our task is to show whether or not coded caching leaks anonymity, so the computational power and memory are not the primary concern at present, and hence Assumption 3.

D. X-D Table and Server Encoding Function

Let $\mathscr{D}$ denote the set containing all possible demand vectors from all K users, and let $\mathscr{X}$ denote the set containing all possible X from $\mathscr{D}$. The X-D table is a two-column table having columns D and X. Column D contains all possible demand vectors in an epoch. Column X contains all bit sequences that are generated by following (1) for the corresponding D's in each corresponding row. Tab. I shows the format of X-D table.

TABLE I: X-D table

D	X
$[d_1, d_2, \cdots, d_K]$	$[X_1, X_2, \cdots]$
$\cdots$	$\cdots$

The server encoding function F_E is used to characterize the mapping relation between $\mathscr{D}$ and $\mathscr{X}$, which is

$$F_E : \mathscr{D} \mapsto \mathscr{X}.$$

Since users file requests are independent from each other in each epoch, there are in all N^K distinct demand vectors. As a result, X-D table has N^K rows.

III. Using X-D Table to Predict D

In this section, we prove Theorem 1 which shows that X and D are one-to-one mapped. Then it becomes straightforward that once the eavesdropper has computed the X-D table, any other users' file requests can be found by looking up D for the corresponding X.

Before introducing Theorem 1, we describe a property between $\mathscr{X}$ and coded subfiles in Lemma 1.

Lemma 1. *If a coded subfile X_i is generated by encoding $t+1$ uncoded subfiles following (1), no user caches all $t+1$ subfiles.*

Proof. After the placement phase, any uncoded subfile is cached by exactly t users. For coded subfile X_i, it is generated by the server to respond to users in group S_i.

Suppose there is a user s who has all the $t+1$ subfiles, the relationship between s and S_i is subject to either of the following two cases:

- **User s is in S_i**
  According to (1), any user in group S_i caches exactly t out of $t+1$ uncoded subfiles and requests the remaining one. As a result, user s who has all the subfiles must not be assigned to group S_i, which means $s \notin S_i$. This case is impossible.
- **User s is not in S_i**
  For any of the $t+1$ uncoded subfile, we have already found t users in group S_i who cache it. Counting user s, at least one uncoded subfile is cached by at least $t+1$ users, which contradicts the placement phase rule.

Neither of the two cases is valid. We prove the lemma. □

From (1) we conclude that X is generated from both D and users cache content Z, so one D corresponds to only one X. Now we prove Theorem 1 which also shows that each X also corresponds to only one D.

Theorem 1. *X and D are one-to-one mapped in the X-D table.*

Proof. We prove the theorem by showing that no two rows share the same X but differ in D.

Suppose there are two rows (D, X) and (D', X') in a X-D table having $X = X'$ but $D \neq D'$. According to (1), any coded subfile $X_i \in X$ is only for users in group S_i. According to Lemma 1, any user who is not in S_i has at most $t-1$ subfiles in X_i, which means it cannot decode X_i to get an uncoded subfile. Therefore, only users in S_i can decode X_i, and each user recovers one uncoded file.

If $X = X'$, all users can only decode the same set of coded subfiles in either X or X'. In other word, users requests are the same, i.e. $D = D'$, which is a contradiction.

Since there are no two rows share X but differ in D, D and X are one-to-one mapped. □

IV. Construct X-D Table From Scratch

In this section, we first introduce the attack algorithm to construct the X-D table. Then an example of conducting the attack is provided. Finally, the complexity of the attack is analyzed and compared against numerical simulation.

A. Algorithm

Stage 1: Interpret the traffic. Stage 1 is to construct function f to map any coded subfile X_i to a set of indexes of subfiles that are used to generate X_i. The mapping can be represented as

$$f : X_i \mapsto [(n_1, j_1), (n_2, j_2), \cdots, (n_{t+1}, j_{t+1})], \qquad (2)$$

where $n_l \in [N]$, $j_l \in [\binom{K}{t}]$, and (n_l, j_l) represents the j_l^{th} subfile of file n_l. For an X that contains $\binom{K}{t+1}$ coded subfiles, X is mapped to $\binom{K}{t+1}$ index sets for each coded subfile.

In real applications, this stage can be bypassed if the indexes of subfiles can be acquired in other ways, such as the header information of the server transmission packets.

In this paper, we assume that each coded subfile corresponds to one and only one set of $t+1$ subfiles. In other words, there are no two distinct sets of $t+1$ subfiles having the same encoding result. This is not always true, especially when subfiles are small. For example, for four subfiles A, B, C and D, if $A = B$ and $C = D$, $(A \oplus B) = (C \oplus D)$ always holds. However, when the size of subfiles is sufficiently large, and files are different from each other, the probability of having one coded subfile corresponding to more than one set of subfiles is negligible. As a result, practically, it's reasonable to make such assumption.

In this stage, the eavesdropper:

1) Stores all the uncoded subfiles into the memory.
2) Tries all the combination of $t+1$ subfiles to generate all possible coded subfiles.
3) Builds up the mapping f between any received X_i and corresponding subfile index set.

Stage 2: Find out the cache content of each user

1) With the help of f, the eavesdropper knows which subfiles have been used to construct the received X. X that contains subfiles from K files is referred to as *anchor sequence*. The demand vector D that corresponds to the anchor sequence is called *anchor demand vector*. The subfile index sets corresponding to an anchor sequence is *anchor set*. To make the description easier, suppose, without loss of generality, that the eavesdropper chooses an anchor sequence X such that the corresponding anchor set contains $\{1, 2, \cdots, K\}$. The corresponding demand vector of this X is then considered to be $D = [1, 2, \cdots, K]$, which labels users with indexes from 1, 2 to K by placing them in the first, second, till K^{th} position of D.
   If user j can decode X_i to get subfile for W_{d_j}, all uncoded subfiles related to X_i but are not from W_{d_j} must be in Z_j.
2) To estimate user j's cache content Z_j, the eavesdropper should find another X' satisfying the following two requirements:
   - All coded subfiles except $\binom{K-1}{t}$ of them in X' are the same as X.
   - The anchor sets for X and X' respectively differ in one file index.

   Bit sequences satisfying the above two requirements are called *support sequences* for user j given anchor sequence X. The demand vectors corresponding to the satisfied support sequences are called *support demand vectors* for user j given anchor demand vector D. The subfile index set corresponding to the support sequence is called *support set*.
3) Let I denote the set of subfiles that are different between X and X'. Let x and x' be the file indexes that show in X and X''s anchor sets, respectively. All the uncoded subfiles from file x and file x' that are used by coded subfiles in I are not in user j's cache Z_j, otherwise, the server shouldn't have sent them to satisfy user j's file request. As a result, the eavesdropper knows $Z_j \cap (W_x \cup W_{x'})$.
4) Repeat steps 2 and 3 for all $j \in [K]$ and $y \in [N]$ until all users' cache contents Z are predicted.

Stage 3: Build X-D table The eavesdropper generates X based on $\mathscr{Z}$ following (1) for all D. The X-D table is then fully reconstructed.

Finally, the eavesdropper knows users cache contents $\mathscr{Z}$, and the mapping relation between X and D. The only remaining uncertainty is the user indexes. In Stage 2 Step 1, user indexes are assigned based on the file index order, therefore, the actual user index is a permutation of the labeled index after Step 1. The power of this attack is that the eavesdropper knows which user requests which file for all the future epochs,

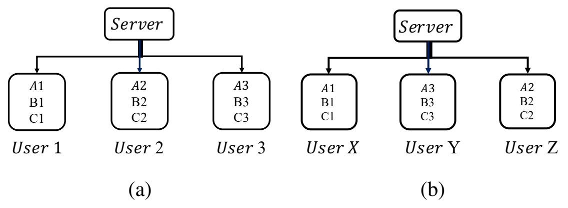

Fig. 2: (a) The network has $N = 3$, $K = 3$ and $M = 1$. Figure shows the cache content after the placement phase. (b) The predicted cache contents of two users after the attack.

TABLE II: Ground truth X-D table.

D	X
[A, A, A]	X1 = A1 ⊕ A2 ‖ A1 ⊕ A3 ‖ A2 ⊕ A3
[A, A, B]	X2 = A1 ⊕ A2 ‖ A3 ⊕ B1 ‖ A3 ⊕ B2
[A, A, C]	X3 = A1 ⊕ A2 ‖ A3 ⊕ C1 ‖ A3 ⊕ C2
[A, B, A]	X4 = B1 ⊕ A2 ‖ A1 ⊕ A3 ‖ B2 ⊕ A3
[A, B, B]	X5 = B1 ⊕ A2 ‖ B1 ⊕ A3 ‖ B2 ⊕ B3
[A, B, C]	X6 = B1 ⊕ A2 ‖ A3 ⊕ C1 ‖ B3 ⊕ C2
[A, C, A]	X7 = A2 ⊕ C1 ‖ A1 ⊕ A3 ‖ C3 ⊕ A2
[A, C, B]	X8 = C1 ⊕ A2 ‖ B1 ⊕ A3 ‖ C3 ⊕ B2
[A, C, C]	X9 = A2 ⊕ C1 ‖ A3 ⊕ C1 ‖ C2 ⊕ C3
…	…
[B, C, B]	X15 = C1 ⊕ A2 ‖ B1 ⊕ A3 ‖ C3 ⊕ B2
…	…
[C, C, B]	X26 = C1 ⊕ C2 ‖ C3 ⊕ B1 ‖ C3 ⊕ B2
[C, C, C]	X27 = C1 ⊕ C2 ‖ C1 ⊕ C3 ‖ C2 ⊕ C3

so the user index mislabeling is not a problem. The correctness of the attack is analyzed in Appendix VI-A.

B. An Example of The Attack

In this example, the network has three users, user 1, 2, and 3, and three files, file A, B, and C. Each user has a cache as large as one file, $M = 1$. In this case, $t = MK/N = 1$, so each file is split into $\binom{3}{1} = 3$ subfiles. The ground truth cache contents after the placement phase is illustrated in Fig. 2a.

After Stage 1, the eavesdropper constructs Table III. Notation ‖ means concatenation.

In Stage 2, the eavesdropper collects X until one anchor set and all the corresponding support sets are collected. In this example, we use the anchor set and support sets shown in Fig. 3 to complete the analysis.

The prediction of user cache contents is as follows. From $\{A2, C1\} \| \{A3, B1\} \| \{B2, C3\}$, the eavesdropper labels the user requesting file A, B, and C as user X, Y, Z respectively. Since X requested A, all the subfiles related to A in the anchor set are for X, such as $\{A2, C1\}$ and $\{A3, B1\}$. X must have subfiles $C1$ and $B1$ to decode $A2$ and $A3$. Since

TABLE III: Server transmission to subfile index set.

X	$f(X)$
X1	{A1, A2} ‖ {A1, A3} ‖ {A2, A3}
X2	{A1, A2} ‖ {A3, B1} ‖ {A3, B2}
X3	{A1, A2} ‖ {A3, C1} ‖ {A3, C2}
X4	{A2, B1} ‖ {A1, A3} ‖ {A2, B3}
X5	{A2, B1} ‖ {A3, B1} ‖ {B2, B3}
X6	{A2, B2} ‖ {A3, C1} ‖ {B3, C2}
X7	{A2, C1} ‖ {A1, A3} ‖ {A2, C3}
X8	{A2, C1} ‖ {A3, B1} ‖ {B2, C3}
X9	{A2, C1} ‖ {A3, C1} ‖ {C2, C3}
…	…
X15	{B2, C1} ‖ {B1, B3} ‖ {B2, C3}
…	…
X26	{C1, C2} ‖ {B1, C3} ‖ {B2, C3}
X27	{C1, C2} ‖ {C1, C3} ‖ {C2, C3}

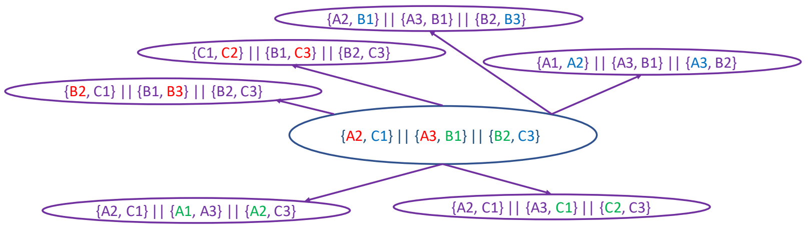
{A2, B1} || {A3, B1} || {B2, B3}
{C1, C2} || {B1, C3} || {B2, C3}
{A1, A2} || {A3, B1} || {A3, B2}
{B2, C1} || {B1, B3} || {B2, C3}
{A2, C1} || {A3, B1} || {B2, C3}
{A2, C1} || {A1, A3} || {A2, C3}
{A2, C1} || {A3, C1} || {C2, C3}

Fig. 3: The anchor set and corresponding support sets chosen by the eavesdropper in Example 2.

the server didn't send $A1$ back, X must have $A1$ in cache. So $Z_A = \{A1, B1, C1\}$. Similarly, $Z_Y = \{A3, B3, C3\}$ and $Z_Z = \{A2, B2, C2\}$. The predicted cache contents of all three users are depicted in Fig. 2b.

In Stage 3, the eavesdropper generates the X-D table based on $\mathscr{X}$. If the eavesdropper labels user X, Y, Z as user 1, 3, 2, the predicted X-D table is the same as the ground truth X-D table.

C. Complexity of The Algorithm

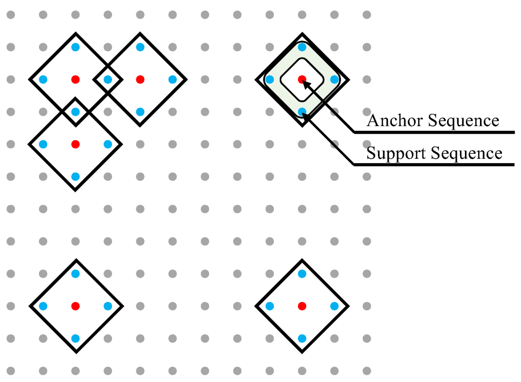

Fig. 4: Points in the graph represent demand vectors in the system. Red points are anchor demand vectors, and each of them is surrounded by blue points, which are support demand vectors.

Stage 1: The task of Stage 1 is to try all combinations of $t+1$ subfiles and build the mapping function f. Since each file is split into $\binom{K}{t}$ subfiles, there are total $N\binom{K}{t}$ subfiles from all N files. The number of combinations of $t+1$ subfiles is $\binom{N\binom{K}{t}}{t+1} \approx (N\binom{K}{t})^{t+1} \approx N^{t+1}K^{t(t+1)}$. One combination of $t+1$ subfiles requires t times XOR operation, and hence the eavesdropper needs to do approximately $t(N\binom{K}{t})^{t+1}$ operations in total. The time complexity is $O(tN^tK^{t^2})$ if this stage cannot be bypassed.

Stage 2: In this stage, the eavesdropper keeps collecting X until one anchor sequence and all corresponding support sequence are captured. The relationship between anchor demand vectors and support demand vectors is shown in Fig. 4.

For the best case (for the eavesdropper), it only needs to collect one anchor sequence and $\binom{K}{1} \cdot (N-1)$ support sequences. However, the probability of encountering the best case is negligible when N and K are large. We turn to estimate the expected number of epochs to process Stage 2, which is E_{bt}.

$$E_{bt} \approx 2N^K. \tag{3}$$

The analysis is in Appendix VI-B.

Stage 3: The eavesdropper now has cache content $\mathscr{X}$. Building the X-D table is simply by implementing 1 to all N^K demand vectors. The overall number of XOR operation would be $N^K \cdot \binom{K}{t+1} \cdot t$, where $\binom{K}{t+1}$ is the number of coded subfiles for one X and t is the number of XOR operations to compute one coded subfile involving $t+1$ uncoded subfiles. The time complexity of this stage is $O(N^K K^t)$.

D. Numerical Simulation

In this section, we compare the actual attack time in Stage 2 to evaluate the performance of our analytical estimation result in (3). The simulation is conducted in Python to simulate a server sending random X and an eavesdropper collecting X. Since our attack doesn't focus on legitimate users behaviors, users are not considered in the simulation.

The server is modeled by a random vector source that generates demand vectors following a uniform distribution. We let the server send demand vectors rather than bit sequences simply because we are in Stage 2, and the mapping function f is regarded to be acquired in Stage 1. The eavesdropper is modeled as a collector who keeps collecting demand vectors until one anchor demand vector and all its corresponding support vectors are collected.

For the first simulation, the number of files is fixed to be $N = 7$, and we change the number of network users K from 2 to 6. The collection process is repeated 20 times for each selection of K. The analytical evaluation following (3) as well as the numerical simulation are illustrated in Fig. 5.

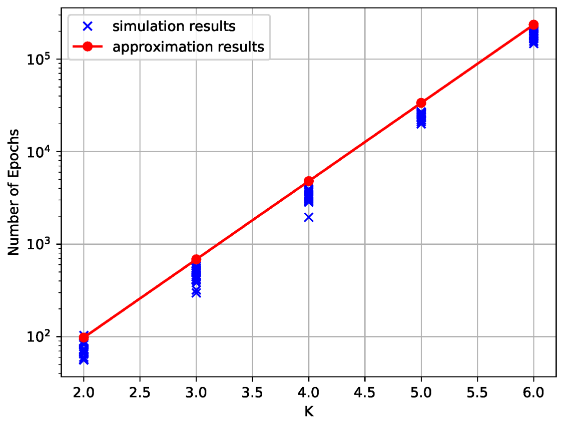

Fig. 5: Number of epochs to complete Stage 2 of the attack when the number of files $N = 7$ and the number of users in the network $K = 2, 3, 4, 5, 6$.

Fig. 5 shows that the time complexity of the Stage 2 process increases exponentially with the number of users. Since we use logarithmic y-axis when plotting, the exponential trend appears linear. As can be seen from the plot, our analytical result captures the exponential factor and aligns well with the simulation results. Interestingly, the estimation performs reasonably good even when N is not much larger than K.

For the second simulation, the number of users is fixed to be $K = 3$ and the number of files N goes from 3 to 20. The collection process is repeated 20 times for each N. The simulation and analytical results are illustrated in Fig. 6.

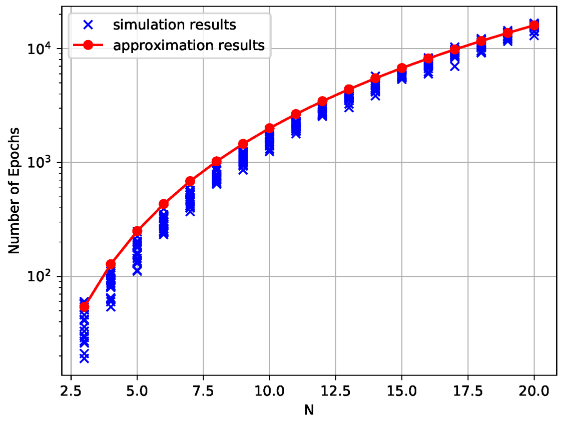

Fig. 6: Number of epochs to finish Stage 2 of the attack when the network has $K = 3$ users, and the number of files N ranges from 3 to 20.

Fig. 6 depicts that the attack time E_{bt} increases as the number of files N in the network increases. The analytical result in (3) depicts E_{bt} to be a power function with respect to N. According to the plot, the analytical result matches the simulation results well.

V. Conclusion

This paper proposes an attack for an eavesdropper to deduce users file requests. The attack makes use of the one-to-one mapping between demand vectors and server transmissions, which we prove that it exists in the well-known coded caching scheme [5], and we provide an algorithm to construct a table to capture the mapping relationship. We thus show that coded caching suffers from complete leakage of anonymity. In future work, we plan to explore methods that can help to protect users privacy/anonymity.

VI. Appendix

A. Correctness of The Attack

Since we assume that all files are public, the eavesdropper is able to get any of the files and build the mapping function f offline in Stage 1. Stage 3 follows (1), so it is also valid. Before proving the correctness of Stage 2, We prove Theorem 2 which illustrates the relationship between an anchor demand vector and a corresponding support demand vector.

Theorem 2. *Demand vector D and D' differ in only one user's file request if and only if bit sequences $X = F_E(D)$ and $X' = F_E(D')$ differ in $\binom{K-1}{t}$ coded subfiles.*

Proof. The theorem has "if and only if" statement, so our proof covers both directions.

1. Necessity

Each coded subfile relates to a group of $t+1$ users, and there are $\binom{K}{t+1}$ groups of such, which means there are $\binom{K}{t+1}$ coded subfiles in one X. Among all the groups, $\binom{K-1}{t}$ of them contain a specific user. Only subfiles that are related to the user who changes her file requrest in D and D' are different. So, X and X' differ in $\binom{K-1}{t}$ coded subfiles.

2. Sufficiency

Suppose X and X' differ in $\binom{K-1}{t}$ coded subfiles, but D and D' differ in s file requests.

According to Lemma 1, coded subfile X_i can only be decoded by users in group i, and each user in group i can only decode out one uncoded subfiles. As a result, no coded subfile can serve the same group but for different users file requests.

When s users change their file requests in D' from D, only coded subfiles related to the remaining $K-s$ users are not changed in X and X'. The number of unchanged subfiles is $\binom{K-s}{t+1}$, and it is a decreasing function w.r.t s. Then, the number of unchanged subfiles is $\binom{K}{t+1} - \binom{K-1}{t} = \binom{K-1}{t+1}$. We can get $s = 1$. This proves the sufficiency of the statement. □

As a result, bit sequences satisfying both the requirements in Stage 2 are support sequences for a given anchor sequence X.

B. Expected Number of Epochs to Build the X-D Table

The computation is divided into three sub-questions:

1) If E_{bt} i.i.d sampled demand vectors are captured with replacement, how many vectors among them are either demand vectors or support vectors?
2) Let's regard one group as a set containing one demand vector and all corresponding support vectors. If E_{bt} number of demand vectors are captured, on average, how many times each group has been chosen?
3) If each group has been chosen y times, how large should y be such that at least one group among all groups collects all the required vectors?

1) Question 1. Relationship between E_{bt} and x: In Stage 2 of the attack algorithm, the eavesdropper needs to collect at least one anchor sequence and all the corresponding support sequences . For a coded caching network having N files and K users, there are N^K different demand vectors. Since only vectors containing K different file indexes are anchor demand vectors, the number of anchor demand vectors N_{acr} is

$$N_{acr} = \binom{N}{K} K!.$$

For one anchor demand vector, the number of corresponding support demand vectors is $\binom{K}{1}(N-1) = (N-1)K$. However, one anchor demand vector can also be another anchor demand vector's support demand vector, and support demand vectors cannot contain more than two repetitive file indexes. So, a support demand vector can only contain either K or $K-1$ different file indexes.

In Section II, we assumed that users file requests are i.i.d sampled from a uniformly distribution. Therefore, each bit sequence $X \in \mathscr{X}$ is equally likely to be captured. On average, when E_{bt} number of vectors are collected, the number of demand vectors having K or $K-1$ different file indexes respectively are

$$E_K = E_{bt} \cdot \frac{N_{acr}}{N^K}, E_{K-1} = E_{bt} \cdot \frac{\binom{N}{K-1}\frac{K!}{2!}}{N^K}.$$

2) Question 2. Relationship between E_{bt} and y: For an anchor demand vector, it is also a support demand vector another group by replacing one of K elements to any other $N-K$ elements not shown in the current vector. For example, $[1,2,3]$ is a support vector for $[r,2,3]$, $[1,r,3]$, or $[1,2,r]$, for any $r \in 1,2,3$. So one anchor demand vector contributes to $K(N-K)+1$ groups. Similarly, a support demand vector having $K-1$ different file indexes, it is a support demand vector for $2(N-K+1)$ by replacing one of the duplicated element to be any other $N-K+1$ not shown file indexes. Then, one support demand vector with $K-1$ different file indexes contributes to $2(N-K+1)$ groups.

When E_{bt} vectors are captured, the number of times each group is expected to be chosen is

$$y = \frac{E_K[K(N-K)+1] + E_{K-1}[2(N-K+1)]}{N_{acr}} \tag{4}$$

3) Question 3. Find out the desired y : For a group containing one anchor demand vector and $(N-1)K$ support demand vectors, finding out the expected number of epochs to cover all the vectors can be modeled as a Markov process shown in Fig. 7 where $\epsilon = 1/[1+(N-1)K]$.

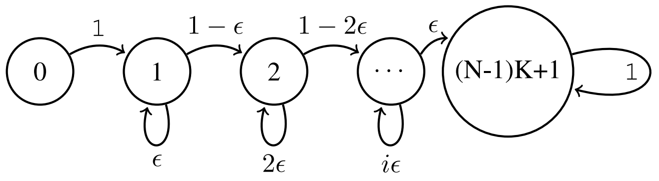

Fig. 7: Markov Chain Model for the Set Covering Problem

In the Markov process model, state $i \in \{0,1,2,\cdots,(N-1)K+1\}$ represents the number of distinct captured vectors . If the state of the system is i and one new vector is captured, the new vector has $i/[(N-1)K+1]$ probability to be a "seen before" vector, i.e. stay in state i, and probability $1-i/[(N-1)K+1]$ to be an unseen vector, i.e. turn to state $i+1$. If the state of the system is $(N-1)K+1$, all the vectors are captured. The expected number of vectors to choose to jump from state i to $i+1$ is $1/(1-i/[(N-1)K+1])$.

Since Stage 2 terminates if any group is fully covered, the task turns into figuring out the corresponding y such that the expected number of fully covered groups is 1.

- If $y < (N-1)K+1$, no group reaches the final state.
- If $y = (N-1)K+1$, $Pr(\text{reach final state}|y)$ is

$$\prod_{i=1}^{(N-1)K+1} \frac{i}{(N-1)K+1} = \frac{\big((N-1)K+1\big)!}{\big((N-1)K+1\big)^{(N-1)K+1}}$$

- For any $y > (N-1)K+1$, $Pr(\text{reach final state}|y)$ is

$$\frac{\big((N-1)K+1\big)!}{\big((N-1)K+1\big)^{(N-1)K+1}} \cdot \Big(\frac{(N-1)K+2}{2}\Big)^{y-(N-1)K-1}$$

Since NK is usually large, we can use the approximate $(N-1)K-1 \approx NK$. By set the expected number of fully covered groups to 1, we get

$$\begin{aligned} y \approx & NK + \frac{NK\log NK - \log(NK)! - \log N_{acr}}{\log NK - \log 2} \\ \approx & NK + \frac{NK\log NK}{\log NK} = 2NK. \end{aligned} \tag{5}$$

Associate (4) with (5), and apply $N-K+1 \approx N$ when N is large, the estimation of E_{bt} is

$$E_{bt} \approx \frac{2N^{K+1}K}{K(N-K+1)+1} \approx 2N^K.$$

References

[1] C. V. N. Index, "Global mobile data traffic forecast update, 2016–2021 white paper," *Cisco: San Jose, CA, USA*, 2017.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 1999, pp. 126–134.

[3] S. Shukla and A. A. Abouzeid, "Proactive retention aware caching," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 2017, pp. 1–9.

[4] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 4, pp. 1029–1040, 2015.

[5] ——, "Fundamental limits of caching," *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, 2014.

[6] A. M. Ibrahim, A. A. Zewail, and A. Yener, "Centralized coded caching with heterogeneous cache sizes," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2017, pp. 1–6.

[7] J. Zhang, X. Lin, and X. Wang, "Coded caching under arbitrary popularity distributions," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 349–366, 2017.

[8] S. Wang, W. Li, X. Tian, and H. Liu, "Fundamental limits of heterogenous cache," *arXiv preprint arXiv:1504.01123*, 2015.

[9] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, "Order optimal coded delivery and caching: Multiple groupcast index coding," *arXiv preprint arXiv:1402.4572*, 2014.

[10] H. Ghasemi and A. Ramamoorthy, "Improved lower bounds for coded caching," *IEEE Transactions on Information Theory*, vol. 63, no. 7, pp. 4388–4413, 2017.

[11] C.-Y. Wang, S. H. Lim, and M. Gastpar, "Information-theoretic caching: Sequential coding for computing," *IEEE Transactions on Information Theory*, vol. 62, no. 11, pp. 6393–6406, 2016.

[12] N. Ajaykrishnan, N. S. Prem, V. M. Prabhakaran, and R. Vaze, "Critical database size for effective caching," in *2015 Twenty First National Conference on Communications (NCC)*. IEEE, 2015, pp. 1–6.

[13] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Characterizing the rate-memory tradeoff in cache networks within a factor of 2," *IEEE Transactions on Information Theory*, vol. 65, no. 1, pp. 647–663, 2018.

[14] A. Sengupta, R. Tandon, and T. C. Clancy, "Fundamental limits of caching with secure delivery," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 2, pp. 355–370, 2015.

[15] F. Engelmann and P. Elia, "A content-delivery protocol, exploiting the privacy benefits of coded caching," in *2017 15th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*. IEEE, 2017, pp. 1–6.

[16] V. Ravindrakumar, P. Panda, N. Karamchandani, and V. M. Prabhakaran, "Private coded caching," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 685–694, 2018.

[17] S. Kamath, "Demand private coded caching," *arXiv preprint arXiv:1909.03324*, 2019.

[18] K. Wan and G. Caire, "On coded caching with private demands," *arXiv preprint arXiv:1908.10821*, 2019.

[19] H. Sun and S. A. Jafar, "The capacity of private information retrieval," *IEEE Transactions on Information Theory*, vol. 63, no. 7, pp. 4075–4088, 2017.

[20] K. Banawan and S. Ulukus, "The capacity of private information retrieval from coded databases," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1945–1956, 2018.

[21] ——, "Asymmetry hurts: Private information retrieval under asymmetric traffic constraints," *IEEE Transactions on Information Theory*, 2019.

[22] R. Tajeddine, A. Wachter-Zeh, and C. Hollanti, "Private information retrieval over random linear networks," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 790–799, 2019.

[23] E. Biham and A. Shamir, "Differential cryptanalysis of des-like cryptosystems," *Journal of CRYPTOLOGY*, vol. 4, no. 1, pp. 3–72, 1991.