Protecting the Long Tail: Transparent Packet Security in Content-Centric Networks

Christopher A. Wood[+]
Department of Computer Science
University of California Irvine
woodc1@uci.edu

***Abstract*—In the Content-Centric Networking (CCN) architecture, content confidentiality is treated as an application-layer concern. Data is only encrypted if the producer and consumer agree on a suitable access control policy and enforcement mechanism. In contrast, transport encryption in TCP/IP applications is increasingly *opportunistic* for better privacy. This type of encryption is woefully lacking in CCN. To that end, we present TRAPS, a protocol to enable transparent packet security and opportunistic encryption for all CCN data. TRAPS builds on the assumption that knowledge of a name gives one access to the corresponding content; otherwise, by design, the content remains encrypted and secure. TRAPS builds on recent advances in memory hard functions and message-locked encryption to protect data in transit. We show that the security of TRAPS is dependent on the distribution of content names and argue that it can be significantly improved if secure sessions are used to transmit small pieces of information from producers to consumers. Our performance assessment indicates TRAPS is capable of providing opportunistic encryption to CCN without significant throughput loss for reasonable packet throughput measurements.**

I. Introduction

Content-Centric Networking (CCN) is an emerging network architecture and protocol for transferring named data between producers and consumers. Unlike traditional IP-based networks in which hosts are directly addressable, CCN treats content as a named and addressable entity that can be moved through the network. Consumers issue requests (interests) for content with a specific name, which are routed towards content producers capable of satisfying the requests. The corresponding content carrying the same name[1] is then sent to the consumer along the reverse path of the request. Routers may choose to opportunistically cache content as it is forwarded towards consumers. This data-centric model decouples content from its place of origin, i.e., the producer, thereby enabling opportunistic content caching within the network. This is done to optimize bandwidth use, reduce latency, and provide effective use of available network resources.

Data privacy and confidentiality are two critical challenges in CCN and related architectures, such as Named Data Networking (NDN) [1]. While structured and semantically meaningful names may aid applications, they reveal a great deal of information to the network. Specifically, names identify precisely what data a consumer desires. If privacy is desired, consumers and producers must agree on a way to obfuscate *application* names into *network* names, i.e., the names encoded in packets. (See [2] for more information about the distinction between application and network names.) This is a very important concept that is worth repeating: applications (and users) yearn for meaningful names, but such utility has a negative effect on privacy.

Now consider content confidentiality. If content is sensitive and must be kept confidential, then it should be properly protected so as to minimize the risk of disclosure to unauthorized parties. Since routers may optionally cache content in transit, encryption is the de facto approach used to protect the information contained in such packets. Much like names, this encryption protection is considered an application-layer concern; if this type of protection is needed, it is assumed the application will implement it accordingly.

Confidentiality is necessarily an application-layer function since it requires some form of authorization. The same is *not true* for privacy. In fact, privacy is now considered a necessary feature for emerging protocols due to growing evidence of large-scale network packet interception and eavesdropping by unauthorized entities [3]. Specifically, pervasive eavesdropping and monitoring is now considered an attack on privacy [4]. To combat these attacks, ubiquitous and opportunistic encryption protocols are being standardized for IP-based protocols such as TCP and DNS, e.g., [5], [6], [7]. Consequently, any viable IP alternative, especially CCN, should deal with the issues of privacy in an equally application-agnostic manner. To the best of our knowledge, there has been no concrete work towards this goal in the CCN community.

In this work we present TRAPS, a mechanism that enables transparent packet security for CCN that, unlike traditional end-to-end encryption mechanisms, does not prohibit packet caching in the network. TRAPS is built on the premise that if one knows an *application* name of content, then it can obtain and decrypt the data. Otherwise, without knowledge of the application name, the data remains encrypted and secure. TRAPS uses application names to create obfuscated network counterparts and encrypt the corresponding content. Thus, TRAPS can be implemented entirely within the network stack, as is done for protocols such as tcpcrypt [6]. Moreover, TRAPS can be easily extended with stronger end-to-end encryption that makes knowledge of an application name insufficient to decrypt content.

Our intended contributions are three-fold:

- The first lightweight, application-transparent "transport" security protocol for CCN.
- End-host network stack modifications necessary to support TRAPS.

[+]Supported by the NSF Graduate Research Fellowship DGE-1321846. Work was done while this author was at PARC.

[1]Technically, content objects can carry no name if the corresponding request specified the hash digest of the content.

ISBN 978-3-901882-94-4

- Analysis of TRAPS security subject to passive eavesdroppers and the performance overhead incurred by end-hosts.

Organization. Section II presents an overview of CCN. Section III motivates the need for TRAPS. Section IV discusses the TRAPS threat model. The cryptographic primitives underlying TRAPS are described in Section V. Section VI presents the TSec protocol and briefly addresses its security claims. The performance overhead introduced by these changes is studied in Section VII. We finally conclude with a discussion of related work in Section VIII and avenues for future work.

II. CCN Overview

This section provides an overview of the CCN architecture and data transfer protocol. It can be skipped without loss of continuity.

A. Architecture and Protocol Description

Unlike IP, which focuses on addressable end-hosts, CCN emphasizes named and addressable data. This subtle difference has strong implications on the underlying communication mechanisms. To obtain content, a consumer issues a request, called an interest, specifying the name of the desired content. The name is a structured, hierarchical name, much like a URI. For example, the name of a file belonging to the UCI Sprout Lab might be named `/edu/uci/ics/sprout/team`. An interest is routed, based on its name, towards an authoritative producer of the content rather than a destination address.

As the interest traverses the network, each router examines the name to determine if it has a copy of the content stored in its Content Store (CS), or cache. If it does, the router transmits the matching content object in reply to the interest. (An interest matches a content object if (a) their names are equal or (b) if the interest carries the hash digest of the content object. Due to (b), content may actually nameless since it is not needed to verify authenticity.) Otherwise, the router records some state derived from the interest (e.g., the arrival interface) in a Pending Interest Table (PIT) so as to provide a reverse path to the requester. Finally the router transmits the interest to the next hop(s) specified in its Forwarding Information Base (FIB). A FIB is a routing table that maps hierarchical name prefixes to outbound interfaces. The router uses longest-prefix-matching (LPM) to index into the FIB to determine the set of possible interface(s) to which the interest should be forwarded. If a router R receives an interest that cannot be satisfied from the cache, and R has already forwarded a previous interest for the same name upstream, it updates the corresponding PIT entry with the new arrival interface. When a content response is then returned to the router, it is forwarded to all downstream interfaces listed in the corresponding PIT entry. In doing so, R may cache the content in anticipation for of a future interest for the same name.

Beyond the pull model that guarantees symmetric interest and content flow, content-centric traffic in CCN has strong security implications. Notably, security is coupled to content rather than its distribution channel. All sensitive content must therefore be encrypted in a meaningful way so as to ensure confidentiality[2]. Content integrity and origin authenticity are typically ensured via digital signatures generated by producers. This is not a requirement, though, as the authenticity of content may be ascertained by checking the cryptographic hash digest of the content that is returned. Specifically, if an interest carries the hash digest of the expected content, the validity of the resultant content can be checked by comparing its hash against that which is provided in the interest.

[2]Other techniques exist for maintaining content confidentiality. We discuss these in Section III.

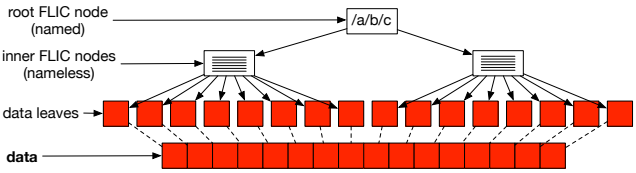

Fig. 1: Sample FLIC tree.

B. Additional Message Constructs

Interests and content objects are the base for different types of messages in CCN. Manifests are a special type of content object used to convey hash digests to consumers. The latter use these hash digests to create interests that carry a name and hash digest for more efficient verification. FLICs are one type of Manifest structure for CCN [8]. A FLIC packet is part of a network-level collection of content objects. Each FLIC node in a collection contains a list of hash digests that are used to construct interests for specific content objects. Each name and hash digest pair is called a *pointer*. (We often use the term *locator* to describe these names since the hash is the real identity of the content.) FLIC nodes may contain or encode pointers to normal content objects (data leaves) or other FLIC nodes, thus creating a DAG structure with a single root and normal content objects as the leaves. Typically, inner-nodes in this DAG are nameless and only the root node has a name. However, the latter is not mandatory. Consumers resolve FLIC root nodes to data leaves by performing an in-order depth-first-search of the pointers in a FLIC node, as shown in Figure 1.

Also, FLICs may contain optional metadata fields such as `DataDigest` to indicate the cryptographic hash of the data contained in *all* of the children pointers[3], as well as `DataSize` to indicate how many data bytes are contained in all of the children pointers. With knowledge of the total number of bytes in a FLIC node, as well as the size of data contained in each node, a consumer can randomly seek to different offsets in the collection.

III. Separating Privacy and Confidentiality

Privacy and confidentiality are pervasive problems in CCN. While they may seem to be orthogonal issues, they often stem from the shared property that some parts of a packet are not protected with encryption unless explicitly done so by the application. Cleartext packet data (and metadata) can reveal details about the producer and intent of the data contained in a content message. It can also leak information about the content requested by a particular consumer. One goal of TRAPS is to provide a transparent mechanism to deter or prevent such types of inference. In this section, we review previous proposals of content confidentiality and problems of name privacy to motivate the need for TRAPS.

[3]This is similar to a Merkle tree digest.

A. Confidentiality Conundrum

The purpose of content confidentiality is to prevent unauthorized parties from accessing protected or sensitive content. This can be done by protecting content with some form of request-based access control scheme [9]. Other (albeit insecure) possibilities include specifying little or no cache time for content objects in the network. Assuming routers are honest, this means that all interests would be routed to the content producer, who could then determine the access rights to content on a per-interest basis. This would require consumers to provide some form of unforgeable identity or authentication token that could be used by the producer to make this authorization decision. The effect of caching on limited availability can therefore be perceived as a (weak) network-layer confidentiality enforcement mechanism.

A more intuitive approach in CCN is to encrypt the content under keys only available to authorized parties. Smetter et al. [10] presented a group-based access control scheme based on the old version of the CCN architecture (CCNx 0.x) [11]. Misra et al. [12] proposed a group-based access control procedure in CCN that uses broadcast encryption [13], [14], [15] to encrypt per-content decryption keys. Ion et al. [16] gave an attribute-based access control scheme for ICNs that applies attribute-based encryption [17], [18] for a similar purpose. Wood et al. [19] presented an access control scheme using proxy re-encryption to personalize cached content objects to each user. Kurihara et al. [20] designed an encryption-based access control framework using CCN manifests to implement, specify, and help enforce access policies. Their framework can be used to instantiate any one of the previously mentioned solutions. NAC [21] gave an alternative to [20] that uses name conventions to construct the names of per-content decryption keys.

While application encryption is by and large the de facto way of protecting content, it is not the only layer at which encryption may be performed. In general, the CCN architecture lacks a form of encryption that (a) does not involve or require any application input and (b) is implemented above the network layer. To the best of our knowledge, there is no protocol for enabling such *transparent* encryption between consumers and producers. We claim that a protocol that would allow this type of encryption is both necessary and timely given that CCN and related architectures are maturing and targeted to replace IP.

B. Pitiful Privacy

Privacy is often an overlooked property in CCN. Many applications rely on well-formed, deterministically generated, and meaningful names to ease the application burden. For example, in the NDN-RTC application [22], names for audio segments are given names of the following format: `<prefix>/ndnrtc/user/<username>/streams/audio0/<bitrate>/...`. This leaks an unnecessary amount of information about the contents and subjects of a conversation. Indeed, overcoming this privacy challenge has deep implications on how names are conveyed to the network. Ghali et al. [23] confirmed (often unstated) intuition that names must be indistinguishable from random strings in order to guarantee some measure of privacy. Their results show that any technique which can decouple application names, such as the NDN-RTC name above, from those which are carried inside packets will help improve privacy. This idea forms the basis of TRAPS.

IV. Threat Model

In recent years, the trend towards transparent, ubiquitous, and opportunistic encryption continues to grow [5], [6]. However, to the best of our knowledge, there is no such protocol analog in CCN or related architectures. Opportunistic encryption is not straightforward when the primary security focus is on data rather than privacy (and the channels through which data flows).

As the name suggests, the goal of TRAPS is to transparently encrypt packets to improve data privacy – *not confidentiality*. Our adversary $\mathcal{A}$ is one which attempts to learn the identity (application name) of encrypted content. $\mathcal{A}$ is active and has the ability to compromise any router in the network. $\mathcal{A}$ can perform any action usually allowed to compromised routers. Unlike standard encryption protocols, such as TLS [24], wherein there is assumed to be a global PKI or possibly shared secrets between trusting parties, we choose to restrict our notion of "transparent" to one in which we rely on neither. As such, the security of TRAPS cannot depend on preconfigured certificates or previously exchanged secrets. (We explicitly omit a design requiring key exchange protocols for TRAPS since it implies a connection. As we will discuss later, a goal of TRAPS is to not break the data-oriented nature of transmission in CCN.) Rather, TRAPS depends on implicitly shared knowledge between consumers and producers. Since there are no cryptographic secrets shared a priori, the security of TRAPS is a function of the amount of work expended by $\mathcal{A}$ to learn this shared information. By default, TRAPS is not intended to be computationally or information-theoretically secure such as is the case with TLS [24]. A powerful enough adversary could break it, but at a cost that is a *design parameter* for the protocol.

V. Cryptographic Preliminaries

TRAPS builds on a couple of new of cryptographic primitives. The first of which are memory-hard functions (MHFs) [25]. A MHF is a function which, on a random access machine, requires $S(\lambda)$ space and $T(\lambda)$ operations to compute, where $S(\lambda) \cdot T(\lambda) \in \Omega(\lambda^2)$ and λ is the security parameter. Optimally, a MHF requires just as much space as it does operations. MHFs are intentionally expensive to compute since they are meant to deter one from computing massive numbers in parallel with custom hardware.

Another primitive we rely upon is so-called convergent or message-locked encryption (MLE) [26]. A MLE scheme is one where the (symmetric) key used to encrypt and decrypt a message is derived from the message itself. In [26], the symmetric key k for message M is derived as $k = H(M)$, where H is a suitable cryptographic hash function, such as SHA256. To encrypt a message M, one then computes a tag $T = H(k)$ and ciphertext $C = \mathsf{Enc}_k(M)$, and then produces the (C, T) tuple. Decrypting and verifying a ciphertext value (tuple) requires one to decrypt the ciphertext, re-derive the tag, and then check for equality. This MLE scheme is deterministic and therefore enables secure de-duplication; identical messages will be encrypted to identical ciphertexts. TRAPS exploits this property for a large class of traffic – specifically, *static* data.

VI. Transparent Packet Security in CCN

In this section we describe the TRAPS protocol and how it sets out to achieve the security goal outlined in Section IV. But first, we must be clear about the requirements of TRAPS:

- R-1 TRAPS should be completely transparent to applications. Neither consumers nor producers should be required to opt in to the protocol. However, the protocol may be exposed to applications to enable stronger security properties.
- R-2 TRAPS should not require consumers or producers to share any cryptographic secrets or perform any sort of key establishment.
- R-3 TRAPS should not break the data-centric and request-based model of CCN. As a result, features such as caching should still work with TRAPS.
- R-4 The security of TRAPS should be a tunable parameter that has a reasonable default and, if desired, can be decided upon by the producer and conveyable to the consumer. Moreover, consumers and producers should be able to opt-out of TRAPS if desired.

The core idea of TRAPS is that implicit knowledge of a name is considered to be a shared secret between a consumer and producer. Consumers know a priori what content they wish to request, whereas producers know what (static) content they provide and are willing to publish. In TRAPS, knowing a name is sufficient to decrypt a packet. Without the name, the data remains encrypted. Thus, the name can be thought of as a sort of password needed to access the underlying data in a packet. From this name, cryptographic secrets can be derived that protect both the name and corresponding data. An eavesdropper who sees a protected request and response learns very little. Moreover, they would have to expend a non-negligible amount of effort, in terms of computation and memory resources, to learn the underlying data.

A. Protocol Overview

At a high level, TRAPS can be viewed as the composition of a password-hashing and encryption algorithm. It uses name obfuscation (via hashing) and content encryption to protect names and data, respectively. Name obfuscation uses a cryptographic (or memory-hard) hash function (F) to map meaningful names to random correspondents. Content encryption uses secret-key cryptographic algorithms ($Enc_k(\cdot)$ and $Dec_k(\cdot)$) for efficiency. We also make use of a key derivation function (KDF), e.g., HKDF [27].

TRAPS is a configurable protocol and accepts the following inputs: Security parameter λ, the obfuscation function $\mathsf{F} : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$, and a salt generation and rotation function, s_G and s_R, respectively. We denote a specific configuration of TRAPS as $\mathsf{TRAPS}(\lambda, \mathsf{F}, s_G, s_R)$.[4]

We now describe TRAPS in more detail. Recall that, to obtain data D with name N, denoted $D(N)$, consumers issue a request (interest) for $D(N)$, denoted $I(N)$. We refer to the i-th component of N as N_i. If $D(N)$ is chunked into n pieces, we denote $D_i(N)$ as the ith piece. The unique identifier for $D(N)$ is its cryptographic hash digest, denoted $D_{ID}(N)$. In this context, N is the *application name* of D. The *network name* $\bar{N}$ carried in the wire-encoded packet and used to forward this request need not be equal to N. However, in standard CCN, $N = \bar{N}$. We use the notation $D(N)$ and $D(\bar{N})$ to refer to the data identified by the given application and network names, respectively. After it is requested, $D(N)$ is carried in a content object message $C(\bar{N})$ with the network name $\bar{N}$. Note that consumers may use different network names $\bar{N}^0$ and $\bar{N}^1$ when requesting $D(N)$, in which case $C(\bar{N}^0) \neq C(\bar{N}^1)$. This can occur if $D(N)$ is uniquely encrypted for each consumer. Conversely, it always holds that if $C(\bar{N}^0) = C(\bar{N}^1)$ then both responses carry the same application data $D(N)$. Recall that it is not a requirement for a content object to carry a CCN name. However, a content object always carries an explicit or implicit identifier that can be matched to a value computed from the corresponding interest. For example, if the content object $C(\bar{N})$ does not carry a name, then its hash digest must match what is provided in $I(\bar{N})$.

[4]When not needed, we omit these parameters for presentation clarity. Also, by default, $s_G = s_R = \perp$, meaning that the salt is the empty string and never changes. If $s_G = \perp$, then it simply returns an empty string upon any input.

1) The consumer application on Cr issues an interest $I(N)$ to the network stack.
2) Cr's network stack computes the obfuscated interest $I(\bar{N}) = \mathsf{Obfuscate}(\mathsf{F}(\cdot), s_G, I(N))$. ($\mathsf{Obfuscate}$ is detailed in Algorithm 1.)
3) The obfuscated interest $I(\bar{N})$ is issued by Cr and routed to P.
4) P's network stack recovers the original name N from $I(\bar{N})$ by looking up the name in a table that maps all obfuscated names to their application representations.[5] P's stack then forwards N to the application.
5) The application returns $C(N)$ to the network stack.
6) P's network stack computes the encrypted content object $C(\bar{N}) = \mathsf{EncryptContent}(C(N))$. If required, $C(\bar{N})$ is then signed.
7) The obfuscated and encrypted content object $C(\bar{N})$ is routed back to the consumer.
8) Cr's network stack computes $C(N) = \mathsf{DecryptContent}(C(\bar{N}))$ and passes it up to the consumer application.

Fig. 2: An summary of the TRAPS protocol.

The complete end-to-end operation of $\mathsf{TRAPS}(\lambda, \mathsf{F}(\cdot), \perp, \perp)$ is shown in Figure 2. Detailed descriptions of each step in the TRAPS protocol are given in Algorithms 1, 2, and 3, and a complete depiction of the TRAPS encryption mechanism is shown in Figure 3. Observe that if a different consumer Cr' application issues an interest for the same name N, and this interest is routed along a path containing a router which has cached the obfuscated content object $C(\bar{N})$, the content object will be returned to Cr' as expected. Since $C(\bar{N})$ contains the name $\bar{N}$ and nonce r, Cr' will be able to decrypt the content object payload before passing up $C(N)$ to the application. Thus, TRAPS can still exploit caches. An alternative strategy would have been for consumers to provide their public key in each interest, similar to the DNSCurve protocol [28]. Producers could encrypt the random content encryption key using this public key and return it in the response. This, however, would not make shared use of caches.

TRAPS is transparent to the network. Only the producer and consumer applications see application names; all network entities, such as routers, deal only with obfuscated network names and encrypted content objects. Moreover, since the translation is deterministic, router processing is unaffected. (Equality on N is functionally the same as equality on $\bar{N}$.) This allows TRAPS to be implemented entirely within the net-

Algorithm 1 Obfuscate($\mathsf{F}(\cdot), s_G, I(N)$)

1: $s = s_G(\mathsf{now}())$
2: $\bar{N} = []$
3: **for** $i = 1 \rightarrow |N|$ **do**
4: $\quad \bar{N} = \mathsf{Append}(\bar{N}, \mathsf{F}(N_1 || \cdots || N_i || s))$
5: **end for**
6: $\mathsf{InterestMap}[\bar{N}] = I(N)$
7: **return** $I(\bar{N})$

Algorithm 2 EncryptContent($C(N), \lambda$)

1: $r \leftarrow \{0,1\}^{\lambda}$
2: $k \leftarrow \mathsf{KDF}(N||r)$
3: $\mathsf{payload} = Enc_k(D(N))$
4: $C(\bar{N}) = (\bar{N}, \mathsf{payload}, r)$
5: **return** $C(\bar{N})$

work stack. Applications may configure TRAPS by choosing λ, F, s_G, and s_R as needed. However, without explicit opt-in, there is a default set of options used for TRAPS: $\lambda = 256$, $\mathsf{F} = \mathsf{SHA256}$, and $s_G = s_R = \perp$.

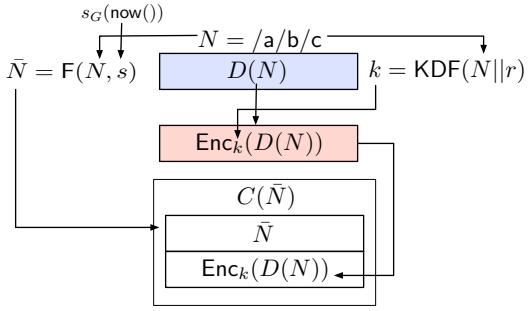

Fig. 3: TRAPS translation and content construction procedure.

B. Static Content

For static content, it is common for a consumer to request N with $D_{ID}(N)$. Doing so requires knowledge of $D_{ID}(N)$ a priori. As discussed in Section II-B, this is typically obtained in a (FLIC) Manifest, denoted $\mathsf{Manifest}(D(N))$. Consumers first fetch the manifest pertaining to a collection of static content objects and then use its contents to subsequently request the constituent chunks by their unique identifier. TRAPS can be applied to each content object chunk in a manifest to encrypt them. However, this is problematic. Since each chunk would have the same name (locator), the same key would be used to encrypt each one. By using the same key to encrypt multiple content objects, we must convey a proper IV or nonce for each one (depending on the mode of operation) such that the encryption remains secure.[6] Lastly, if the encryption key is based on N, then each encrypted chunk becomes eternally bound to N. This prevents de-duplication that might arise as nameless chunks move around the network under different routable prefixes.

Therefore, to support de-duplication even when N (the locator) changes, the encryption key must be based on something else. To address this problem, we turn to MLE. In particular, we let the encryption key for a chunk be derived from its application data contents and, optionally, a name

[6]Reusing a key and nonce pair with AES-GCM has disasterous consequences, see e.g., [29].

Algorithm 3 DecryptContent($C(\bar{N})$)

1: $I(N) = \mathsf{InterestMap}[\bar{N}]$
2: $k \leftarrow \mathsf{KDF}(N||r)$
3: $\mathsf{payload} = Dec_k(C(\bar{N}).\mathsf{payload})$
4: $C(N) = (N, \mathsf{payload})$
5: **return** $C(N)$

as well. This name can be N or $\bar{N}$, depending on how restrictive are the requirements for access to $D(N)$. Let $\mathsf{KeyGen}(\cdot,\cdot)$ be a function that takes as input $D(N)$ and optional name $N \in \{\perp, \mathbb{N}\}$ to compute an encryption key k. A key $k = \mathsf{KeyGen}(D(N), \perp)$ will encrypt the content to be accessed with any locator and can be moved anywhere in the network. (This is the default configuration.) A key $k = \mathsf{KeyGen}(D(N), N)$ will bind k to the locator N. Lastly, a key $k = \mathsf{KeyGen}(D(N), \bar{N})$ binds k to the ephemeral, obfuscated locator that is observed in the network. The exactly derivation mechanism to be used is a design choice for the producer stack, provided that they convey this decision to consumers. By default, the $k = \mathsf{KeyGen}(D(N), \perp)$. With k, the producer can then encrypt each chunk $D_i(N) \in D(N)$ in such a mode that permits random access decryption. (This is necessary so that chunks may be received out of order or in partial without preventing their use.) For integrity reasons, the producer should also generate a tag as outlined in Section V. This tag can be included in the content object in place of a signature.

The final part of this approach is to convey $D_{ID}(N)$ to consumers, since a consumer cannot derive a hash for content to which it does not have access. For this, we use $\mathsf{Manifest}(D(N))$. The FLIC manifest already carries the hash of $D(N)$ as additional metadata. Thus, by retrieving a FLIC manifest, a consumer can then derive the decryption key(s) for the constituent chunks. Figure 4 shows this manifest-based construction visually.

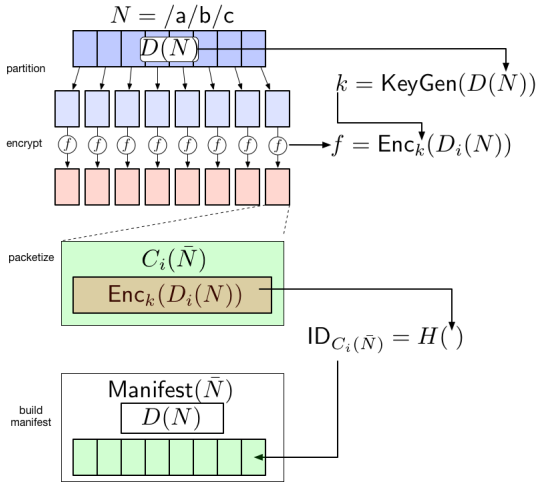

Fig. 4: TRAPS FLIC manifest construction procedure.

C. Dynamic Content

The protocol described in Sections VI-A and VI-B assumes that all content is static, i.e., not generated dynamically or on-demand. This enables the producer to precompute the data to

map network names to their application counterparts. This is not possible for dynamic content due to the preimage-resistant nature of F.

There are (at least) two ways to support dynamic content with TRAPS. The first is to additionally encrypt N, with application-specific components, under the producer's public key pk_P. This encrypted name could be included in the payload of the interest so that routing still occurs on $\bar{N}$. When interests of this form arrive at the producer, the latter can simply decrypt the payload using the corresponding secret key sk_P to recover N. To acquire pk_P without first talking to P, consumers could use a key-resolution service such as the CCN-KRS system proposed in [30]. A second alternative is for consumers to establish a session using, e.g., the CCNx-KE protocol [31], over which to transfer encrypted dynamic content. Note that this approach requires consumers and producers to opt-out of TRAPS so that name transformations are not inadvertently applied.

D. Discovery and Interoperability

TRAPS relies on consumers knowing the name of content they wish to access. For nameless objects, if the hash digests of content are unpredictable, then the encrypted content objects are protected from eavesdroppers. In section VI-B, we assumed that manifests carrying these hash digests were themselves encrypted using TRAPS. Therefore, if the manifest name were predictable, then the hashes could be retrieved. This method of protecting manifests is necessary for backwards compatibility, since it places no additional burden on the network.

However, to improve this, manifests could be protected in a number of different ways. First, a secure session established via CCNxKE [31] could be used to transfer the manifest. This would prevent passive eavesdroppers between the consumer and producer from learning the hash digest of nameless content objects in the manifest so as to decrypt them. Alternatively, if access control schemes such as IBAC [9] are used, wherein the consumer and producer share keying material that can be used to encrypt names, then the manifest could be fetched with an encrypted name. This effectively restricts access to the manifest, and all content objects contained therein, to only authorized consumers. While using a session or IBAC to fetch manifests would work, it introduces extra complexity into applications that is otherwise not needed by TRAPS.

E. Dictionary Attacks

When $s_G = \perp$, TRAPS is susceptible to dictionary attacks [32] since the only secret information in the protocol is N, the application name, which is known by all consumers who request content.[7] A dictionary attack is where an adversary precomputes hash digests from a dictionary or list of popular inputs so that s/he can easily reverse these values. The first dictionary attack deterrence built into TRAPS is the use of salts, generated via s_G, to obfuscate names.

There are several possible options for s_G. The first is for s_G to return $s \xleftarrow{\$} \{0,1\}^\lambda$, and to rotate (update) this value every s_R seconds. The problem with the salt is consumer synchronization: how does a consumer get the salt before issuing $I(N)$? If the salt is *explicit*, then the producer must generate and publish the salt for consumers to obtain. Specifically, let N_s be the salt name published with the obfuscated name $\mathsf{Obfuscate}(N_s)$. Cr could issue an interest for N_s to obtain the salt s, and then use s to subsequently derive the names of desired content. For all subsequent interests, Cr could then use s in the obfuscation procedure.

Therefore, we recommend a simpler input for the salt: time. Here, s_G is equal to the current time epoch divided by s_R. For example, if the current time epoch (UNIX time) is 1483921358 (01/09/2017 @ 12:22am UTC), and s_R is 10000, then $s = 1483921$. Consumers and producers share knowledge of time and can be assumed to be in sync within some loose margin of error. As the granularity of time (s_R) decreases, the probability that it is shared between consumers and producers increases. Of course, since time is predictable, it is possible for a powerful attacker to pre-compute obfuscated names with future versions of time. This could be mitigated by mixing both time and a producer-provided salt on a regular frequency. For example, producers could publish a new salt every day or week, which would then be required when performing the F computation.

Another deterrence built into TRAPS is to use MHFs as F. These dampen the efficacy of offline dictionary attacks while adding more online computational and memory overhead to consumers. This performance tradeoff is assessed in Section VII.

F. Security Analysis

The security of TRAPS dissappears as information (names and data) become predictable. This is because trial decryption driven by dictionary attacks are possible. However, there is one crucial element of TRAPS that limit the efficacy of trial decryption attacks: Since each content key is derived from a fresh and random nonce, $\mathcal{A}$ cannot attempt trial decryption until *after* it observes an encrypted content object. This means dictionary attacks targeting the content must be *online*. Given the amount of traffic that is sent on a network, this means that $\mathcal{A}$ has to either have a tremendous amount of computational resources available or must be selective in which content it attempts to decrypt. Both conditions make less popular or predictable content, i.e., content in the "long tail" of the popularity distribution, intuitively more safe.

Another avenue for attack is through the obfuscated name via an offline dictionary attack. The cost of the dictionary attack is controlled by the parameters of F. To illustrate this effect, let the security of F be defined by parameters space and time parameters s and t. Let $\mathbb{N}$ be a set of names from which the attacker will sample. Moreover, let $\mathcal{D}(\mathbb{N})$ be the distribution of these names such that $\mathcal{D}(N)$ for $N \in \mathbb{N}$ is the probability that N is selected when sampled from $\mathbb{N}$. The best dictionary attack is one where the attacker proceeds as follows. Simply traverse $\mathcal{D}(\mathbb{N})$ in descending order by probability and compute $\mathsf{Obfuscate}(\cdot, N)$ for each N until the target value is found. We may estimate the complexity of this attack as follows. Let N^* be the actual name represented by a packet with the obfuscated name $\bar{N}$. Let $C(s,t)$ be the cost of computing F given parameters s and t. Now, assuming N^* is the k-th most popular name in $\mathcal{D}(N)$, the attack will then

[7] Using deterministic public-key encryption for name obfuscation would serve no better than the hash function. Since adversaries would also have access to the producer's public key, and could therefore compute the obfuscated names as easily as regular consumers.

require (at most) k computations of Obfuscate and therefore cost $k \times C(s,t)$. This leads to the following minimum work bound for these offline dictionary attacks.

Definition 1: Given a space and time parameters s and t, as well as a set of names $\mathbb{N}$ with distribution $\mathcal{D}(\mathbb{N})$ whose expectation is $E(\mathbb{N})$ and PMF is $f(\cdot)$, the average amount of work required to conduct a dictionary attack on one name $N \leftarrow \mathbb{N}$ is $(1 - f(E(\mathbb{N}))) \times C(s,t)$.

If $\mathcal{D}(\mathbb{N})$ is the uniform distribution with 10 elements, then the expected cost is $5 \times C(s,t)$. Similarly, if $\mathcal{D}(\mathbb{N})$ is the Zipf distribution with 10 elements, whose expected value is $\frac{H_{10,\alpha-1}}{H_{N,\alpha}}$, then the expected cost is $(1 - \frac{H_{10,\alpha-1}}{H_{N,\alpha}}) \times C(s,t)$. The cost obviously scales (linearly) as the set of names increases.

VII. TRAPS Analysis

A. End Host Overhead

Computationally, TRAPS only introduces overhead at consumers and producers. Routers and other network entities are unaffected by interests and content object messages using TRAPS obfuscation and encryption. Therefore, to assess the performance of TRAPS, we quanitfy the overhead incurred by consumers and producers. This overhead is divided into four parts: (1) interest obfuscation, (2) interest de-obfuscation, (3) content encryption, and (4) content decryption. Each of these four procedures adds a certain amount of processing overhead to each message as it traverses the consumer and producer network stacks. All experiments were performed on a workstation with a 2.8 GHz Intel Core i7 CPU and 16GB of 1600 MHz DDR3 RAM running Ubuntu 14.04.

To quantify this processing overhead, we implemented each of the operations in C on top of the Libccnx and Libparc [33] libraries. For input, we used the Unibas dataset from the The Content Name Collection [34], which contains *unique* URLs submitted by users to URL shortener websites. We converted these URLs into a CCN-compatible name format. For example, the URL `http://www.domain.com/file.html` was converted into the CCN name `/com/domain/file.html`.

This dataset does not provide any information about the corresponding content object sizes, so we randomly generate the sizes between the range of 1.5KB and 9KB. For efficiency, we use ChaCha20+Poly1305 [35] for the authenticated encryption algorithm to protect content objects. Randomness for nonce generation is drawn from `/dev/urandom` (it does not block). For comparison we use both SHA256, which is *not* a memory-hard function, and Argon2 [36], which is a very recent MHF, as the name obfuscation function F. The code for which is available at [37].

The results from this experiment are shown in Figure 5. The name de-obfuscation procedure, which is simply a hash table lookup, is negligible compared to the remaining steps. Request obfuscation is the most expensive step in the protocol since it requires the most computation and is necessarily linear in the length of the input name. Content encryption and decryption perform moderately better; encryption involves more work since it must first sample randomness necessary to generate the random nonce. Overall, the worst-case time for a single step in TRAPS on this machine is approximately 60μs, which is well below the network I/O bottleneck and therefore within reason.

B. Consumer Throughput

Of all the computations required in TRAPS the name obfuscation step is the most expensive and thus the throughput bottleneck. With the use of memory hard functions in lieu of traditional hash functions, the throughput ceiling is lowered even further. Thus, applications must take care to not use obfuscation functions that disrupt normal QoS. This has two implications on the function used: (1) the computational overhead must never exceed the network overhead and (2) the number of possible hash functions per second should always exceed the number of packets *sent* per second. This suggests two strategies in selecting the obfuscation function parameters. Let R be the maximum number of bytes per second sent by a client C. Let L be the minimum link MTU from C to each producer $P_1, \ldots, P_n$. Let T_{min} be the minimum RTT between C and any P_i. Finally, let T_{obf} be the (worst-case) time required to obfuscate a name in a single packet.

Based on the previous two conditions for throughput, it must be the case that

$$\frac{PT_{obf}}{L} < 1 \tag{1}$$

and that $T_{obf} \leq T_{min}$. Equation 1 states that the total time consumed by hashing all required packets in a second does not exceed 1 second. Otherwise, the system would be unstable since the client could not keep up with the desired packet transmission rate.

Equation 1 also places an upper bound on the number of packets that can be sent every second. Given T_{obf} and assuming an approximate value of $L = 1500$B, we can compute an upper bound on P to satisfy this inequality. That is, we can find the largest P such that $P < \frac{L}{T_{obf}}$. To estimate this capacity, we profiled SHA256 and Argon2 to measure the expected throughput under a variety of configurations. The results are plotted in Figure 6. As expected, we can achieve packet throughput rates on the order of 10^8 per second with SHA256, but with a $m = 25$ (2^{25} KiB) memory cost for Argon2 this drops down to 10^5 per second.

We may also write Equation 1 as

$$T_{obf} < \frac{L}{P} \tag{2}$$

and, for reasonable values of P, find the obfuscation function parameters that bring it close to this upper bound. For example, assume $P = 100$Mbps and $L = 1500$B. Then, $T_{obf} < 15\mu$s. This is well within the bounds when using SHA256 as the obfuscation function. However, this is not the case for Argon2. For P values of 2Mbps and 128Mbps, the parameters $(8, 26, 1)$ and $(1, 8, 1)$, respectively, can be used to meet the throughput criteria. Larger values for m indicate that more memory is used for the function, which is the ultimate limiting factor in its performance as well as the primary factor in its security (see [36] for more details). For comparison, Netflix Ultra-HD quality streaming requires a throughput of 25 Mbps [38]. Based on this assessment, TRAPS can certainly meet this requirement.

VIII. Related Work

[39] attempted to enable producer privacy by mixing sensitive information in with fake or cover content. The goal was to mask the real content returned in response to an interest.

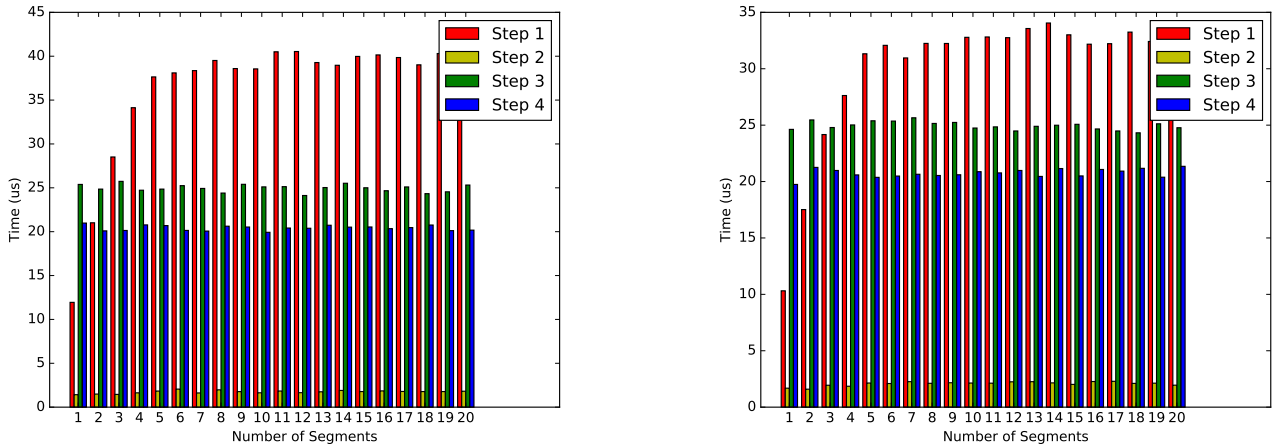

Fig. 5: Overhead of each step of TRAPS using different obfuscation functions. The left plot uses Argon2d with parameters $t = 6$ and $m = 12$ (2^{12} KiB) and the right plot uses SHA256.

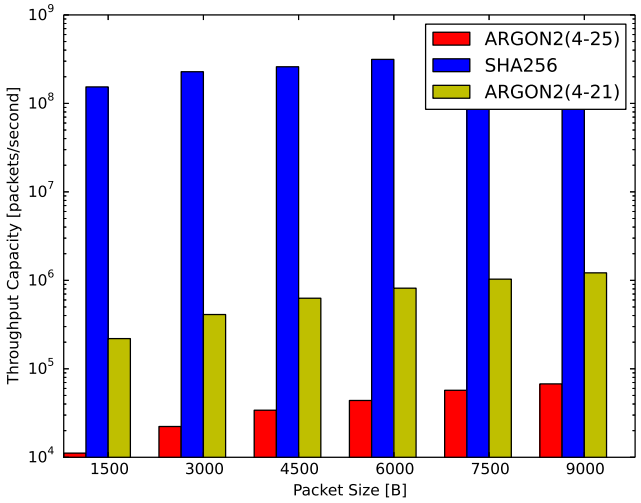

Fig. 6: Throughput capacity when using Argon2 with $t = 4$ and $m \in \{21, 25\}$ and SHA256 as F.

This technique is problematic since it requires mandatory and honest producer participation in order for their privacy guarantees to be achieved. Furthermore, it placed an unnecessary storage and computational burden on producers and consumers. Dibenedetto et al. [40] focused on consumer and producer anonymity rather than data privacy. Their system, called ANDaNA, follows the Tor [41] model for onion-encrypting (and decrypting) packets in transit between a source and destination. However, ANDaNA is an application-layer service that is not transparent to the network. Moreover, it involves online public-key cryptographic operations to forward packets. Ghali et al. [23] showed that privacy in CCN is nearly futile unless there is some notion of a shared secret between consumers and producers. Their work supports the main security argument of TRAPS: unpredictable names yield private and encryption packets.

CCN confidentiality is a more popular problem in the literature [10], [12], [16], [19], [20], [21]. All of the approaches involve some form of application-layer encryption and key management technique using a variety of different cryptographic algorithms to enforce access control, e.g., broadcast encryption and proxy re-encryption.

Contrary to CCN, the problems of privacy and confidentiality are well studied in the IP landscape [42], [5], [6], [7], [28], [41]. Protocols such as IPSec [42] and tcpcrypt [5] exist to protect individual packets from eavesdroppers by encryption performed deep in the network stack. They both involve some handshake protocol (e.g., tcpcrypt key establishment or IPSec IKE) to establish a secure channel with a shared cryptographic key. In effect, IPSec and tcpcrypt encrypt all data above the network and transport layer of the TCP/IP stack, respectively, which protect the contents of each individual packet sent between a client and server. Although IPSec is older, tcpcrypt is likely to obtain more widespread adoption due to (a) its simplified key establishment protocol that extends the standard 3-way TCP session establishment protocol by a single additional message, and (b) there is greater incentive to encrypt at higher layers in the protocol stack.

Per-packet encryption via tcpcrypt and IPSec provides confidentiality and privacy for communication with well-known hosts; only source and destination addresses are visible in cleartext to eavesdroppers. However, discovering the addresses of these hosts via the DNS often reveals information that breaks these privacy assurances. DNSCurve [28] and DNS-over-TLS [7] are two similar approaches to this leakage problem which seek to provide DNS query privacy by encrypting the contents of each request. These approaches are different from those that use Tor[41] to hide the location or origin of DNS queries since they only hide the query contents.

IX. Conclusion

We presented TRAPS, a protocol for enabling transparent packet security in CCN. It is based on the idea that only those

with knowledge of a name can ask for the data and decrypt it upon receipt. It is secure up to chosen name distributions. For predictable names which are subject to dictionary attacks, the attacker's job is made difficult by (a) using MHFs to make offline exhaustive search expensive and (b) using per-request nonces to prevent offline dictionary attacks of content. We presented the design of the protocol and discussed its security. We then provided a comprehensive assessment of its performance impact on end-hosts. Our results indicate that for reasonable measures of packet throughput, TRAPS is a simple and yet powerful way to deter widespread network eavesdropping attacks on CCN.

REFERENCES

[1] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[2] Cesar Ghali, Gene Tsudik, and Christopher A Wood. Network names in content-centric networking. In *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*, pages 132–141. ACM, 2016.

[3] CNN. Latest nsa leaks point finger at high-tech eavesdropping hub in uk, 2013. http://www.cnn.com/2013/12/20/world/europe/nsa-leaks-uk/.

[4] Stephen Farrell and Hannes Tschofenig. Pervasive monitoring is an attack. 2014.

[5] Andrea Bittau, Michael Hamburg, Mark Handley, David Mazieres, and Dan Boneh. The case for ubiquitous transport-level encryption. In *USENIX Security Symposium*, pages 403–418, 2010.

[6] Andrea Bittau, Michael Hamburg, Mark Handley, David Mazieres, and Dan Boneh. Simple opportunistic encryption. 2014.

[7] Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. Connection-oriented dns to improve privacy and security (extended). *USC/Information Sciences Institute, Tech. Rep. ISI-TR-2015-695, Feb*, 2015.

[8] Christian Tschudin and Christopher Wood. File-Like ICN Collection (FLIC). Internet-Draft draft-tschudin-icnrg-flic-01, Internet Engineering Task Force, July 2016. Work in Progress.

[9] Cesar Ghali, Marc A Schlosberg, Gene Tsudik, and Christopher A Wood. Interest-based access control for content centric networks. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 147–156. ACM, 2015.

[10] Diana K. Smetters, Philippe Golle, and J. D. Thornton. CCNx access control specifications. Technical report, PARC, July 2010.

[11] CCNx. http://ccnx.org/.

[12] Satyajayant Misra, Reza Tourani, and Nahid Ebrahimi Majd. Secure content delivery in information-centric networks: Design, implementation, and analyses. In *Proc. ACM SIGCOMM ICN 2013*, pages 73–78, August 2013.

[13] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Proc. CRYPTO 2005*, pages 1–19, August 2005.

[14] Wen-Guey Tzeng and Zhi-Jia Tzeng. A public-key traitor tracing scheme with revocation using dynamic shares. In *Proc. PKC 2001*, pages 207–224, February 2001.

[15] Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In *Proc. FC 2000*, pages 1–20, February 2000.

[16] Mihaela Ion, Janqing Zhang, and Eve M. Schooler. Toward content-centric privacy in ICN: Attribute-based encryption and routing. In *Proc. ACM SIGCOMM ICN 2013*, pages 39–40, August 2013.

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

[18] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proc. ACM CCS 2006*, pages 89–98, October–November 2006.

[19] Christopher A. Wood and Ersin Uzun. Flexible end-to-end content security in CCN. In *Proc. IEEE CCNC 2014*, January 2014.

[20] Jun Kurihara, Christopher Wood, and Ersin Uzuin. An encryption-based access control framework for content-centric networking. *IFIP*, 2015.

[21] Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. Name-based access control. *Relatório Técnico TR NDN-0034, University of California, Los Angeles, Los Angeles*, 2015.

[22] Peter Gusev and Jeff Burke. Ndn-rtc: Real-time videoconferencing over named data networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 117–126. ACM, 2015.

[23] Cesar Ghali, Gene Tsudik, and Christopher A Wood. (the futility of) data privacy in content-centric networking. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, pages 143–152. ACM, 2016.

[24] Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[25] Colin Percival. Stronger key derivation via sequential memory-hard functions. *Self-published*, pages 1–16, 2009.

[26] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 296–312. Springer, 2013.

[27] Hugo Krawczyk and Pasi Eronen. Hmac-based extract-and-expand key derivation function (hkdf). Technical report, 2010.

[28] Daniel J Bernstein. Dnscurve: Usable security for dns, 2009.

[29] Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. Nonce-disrespecting adversaries: Practical forgery attacks on gcm in tls. 2016.

[30] Priya Mahadevan, Ersin Uzun, Spencer Sevilla, and JJ Garcia-Luna-Aceves. Ccn-krs: a key resolution service for ccn. In *Proceedings of the 1st international conference on Information-centric networking*, pages 97–106. ACM, 2014.

[31] Christopher Wood, Ersin Uzun, and marc.mosko@parc.com. CCNx Key Exchange Protocol Version 1.0. Internet-Draft draft-wood-icnrg-ccnxkeyexchange-01, Internet Engineering Task Force, October 2016. Work in Progress.

[32] Benny Pinkas and Tomas Sander. Securing passwords against dictionary attacks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 161–170. ACM, 2002.

[33] CCNx distillery. https://github.com/parc/CCNx_Distillery. Accessed: May 14, 2016.

[34] The content name collection. http://www.icn-names.net/. Accessed: April 8, 2016.

[35] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, October 2015.

[36] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302. IEEE, 2016.

[37] TRAPS performance evaluation code. https://github.com/chris-wood/tsec-performance.

[38] Netflix. Internet Connection Speed Recommendations. https://help.netflix.com/en/node/306.

[39] Somaya Arianfar, Teemu Koponen, Barath Raghavan, and Scott Shenker. On preserving privacy in content-oriented networks. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, pages 19–24. ACM, 2011.

[40] Steven DiBenedetto, Paolo Gasti, Gene Tsudik, and Ersin Uzun. Andana: Anonymous named data networking application. *arXiv preprint arXiv:1112.2205*, 2011.

[41] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[42] Stephen Kent and Randall Atkinson. Security architecture for the internet protocol, 1998.