Demo Proposal for IEEE LCN 2014
Cryptographically-Curated File System (CCFS):
Secure, Inter-operable, and Easily Implementable Information-Centric Networking

Aaron D Goldman*, A. Selcuk Uluagac**, John A Copeland*
*Communications Systems Center (CSC) **Communications Assurance and Performance (CAP) Group
{goldman, selcuk}@gatech.edu {john.copeland}@ece.gatech.edu
School of Electrical and Computer Engineering
Georgia Institute of Technology Atlanta, Georgia 30332–0250

Abstract
Cryptographically-Curated File System (CCFS) proposed in this work supports the adoption of Information-Centric Networking. CCFS utilizes content names that span trust boundaries, verify integrity, tolerate disruption, authenticate content, and provide non-repudiation. Irrespective of the ability to reach an authoritative host, CCFS provides secure access by binding a chain of trust into the content name itself. Curators cryptographically bind content to a name, which is a path through a series of content objects that map human meaningful names to cryptographically strong content identifiers. CCFS serves as a network layer for storage systems unifying currently disparate storage technologies. The power of CCFS derives from file hashes and public keys used as a name with which to retrieve content and as a method of verifying that content. In this demo, we present our prototype implementation to show CCFS exposed as a file-system and a webserver.

Keywords: *Inter-operable Heterogeneous Storage, Content Centric Networking (CCN), Delay Tolerant Networking (DTN), Information Centric Networks (ICN), Name Oriented Networking (NON), Cryptographically Curated File System (CCFS), Self-Certifying File Systems, FUSE.*

I. Significance of the Demo

The power of CCFS derives from the dual usage of file hashes and public keys, as both a name with which to retrieve content, as well as, a method of verifying the content. CCFS simplifies the current dominant networking paradigm which includes three distinct hierarchies for naming, trust, and retrieval into only one naming hierarchy. The dominant naming model is the DNS; the trust model is the X.509 certificate chain; and the retrieval model is CDN (content delivery networks). In CCFS, the naming convention serves as all three: a unique identifier to reference content, as a trust model to verify the integrity of the content and its inclusion in a curated collection, and as a sequence to be used to retrieve the content. CCFS maps between two types of names. One is the hierarchical arbitrarily-chosen name that is required by humans and by many applications. The other is a flat self-certifying unique name that is capable of being used by a name oriented network. CCFS provides this mapping in a cryptographically secure way by using hashes and signatures. This mapping will cause name oriented networks to be compatible with existing file-systems as well as with the DNS paradigm. This demo will show the implementation of the CCFS paradigm to the participants of the LCN 2014 Conference.

II. CCFS Architecture

CCFS is a trust model based on a distributed system of content objects that can be cryptographically referenced and authenticated by their hashes and signatures in a hierarchy defined by chains of trust. The chain of trust in CCFS results from the curator choosing which other curators to include in their namespace as opposed to X.509 where content hosts choose the authority that will certify their site. Currently, validating a message means authenticating the channel. Under CCFS to validate a message means to authenticate the content and the curator. The CCFS architecture is described below by explaining the components and the procedures of CCFS.

A. Components of CCFS

The CCFS model requires the use of two cryptographic primitives (hash and signature), four types of content objects, and two lookup services. The construction of these components is described below. Content Object Types: CCFS utilizes four types of Content Objects, as described in Table I.

TABLE I
CCFS CONTENT OBJECT TYPES

TYPE	PURPOSE	CONTENT FORMAT
Blob	Static; contains data	{ByteArray}
List	Static; Exclusive map from: next name segment to: HID, Type	{[NameSegment, ObjectHash, ObjectType] }
Commit	Versioned; Exclusive map from:HKID to:List's HCID	{ListHCID, Version Number, ParentHCID, Signature, HKID}
Tag	Versioned; Non-exclusive map from:HKID, next name segment to: HID,Type	{ObjectHash, ObjectType, NameSegment, Version Number, ParentHCID, Signature, HKID}

The "BLOB" object is a data container with no inherent structure. The BLOB is the content of interest requested by the user. A BLOB is referred to by the Hash of Content Identifier (HCID).
The "LIST" object is a table used to look up a human readable name to obtain the type and hash (HID) of the content to which it refers. The HID is the cryptographic name of the Content Object. A LIST is a specific type of blob and, like other blobs, can be referred to by its Hash of Content Identifier (HCID). The exclusive use of the BLOB and LIST types, would allow users of CCFS to identify static content only.
Any modification of the BLOB or LIST creates a new HCID and requires updating all references to the content. To accommodate identifying collections of dynamic content (content that changes over time) we define two additional Content Objects: a COMMIT and a TAG. These content objects are used to refer to updatable content and are referenced by an HKID, Hash of Key IDentifier (HKID). A TAG is similar to a COMMIT, but is used to indicate a specific version of a single item within a Domain, a collection that versions items in the collection independent of other items in the collection.

B. CCFS Procedures
The main operations performed with CCFS are to retrieve and verify content (R1-R8) and to curate content (C1-C9). To retrieve the target content, the system follows the iterative steps shown in Figure 1. and simplified in Figure 2. The steps to perform these operations are listed below the figures.

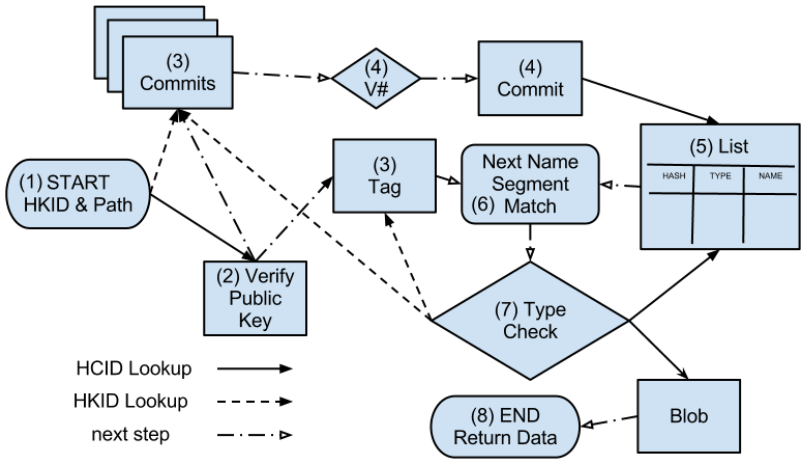

Fig 1. CCFS Retrieval Flow Chart

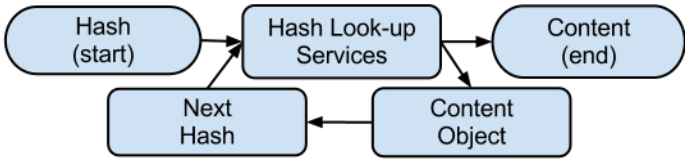

Fig 2. CCFS Iterative Retrieval Process

Retrieval R1 START: Input HKID and Human Readable Path to Content
R2 PUBLIC KEY VERIFICATION: Obtain public key for HKID; If key hashes to HKID, then key is valid
R3 LOOK UP COMMIT/TAG: Retrieve latest version Commit and Tag that contains HKID
R4 VERSION CHECK: Verify signature on most recent version
R5 LOOK UP LIST: Extract HCID from the Commit or Tag; Retrieve List for HCID
R6 NAME SEGMENT MATCH: From List, extract HID and Type associated with next Name Segment
R7 TYPE CHECK: If Type is Commit, use HKID to find next Commit; Check version. If Type is Tag, use HKID to find Tag with next Name Segment; Check Version. If Type is List, use HCID to find List; if HCID verifies Content Object, go to next Name Segment. If Type is Blob, use HCID to find Blob, then if HCID verifies Content Object, return content.
R8 END: If verification has succeeded, return data with successful status code

The curation process is an extension of the retrieval process and the procedures are listed below:
C1 Curator selects content and chooses a Human Readable Name, for example, pathntonfile.txt
C2 Follow the Name path as if retrieving the content
C3 Keep track of Commits and Tags to determine last Commit or Tag (closest to the end of the name), and the Lists that follow the Commit or Tag
C4 Publish the content as a Blob to storage
C5 Generate or Update List Object with HCID of the new Blob
C6 Repeat step 5 until a Commit or Tag precedes the revised list
C7 Update Commit or Tag with the new List, signing with Private Key associated with the HKID
C8 Publish the Commit or Tag Object to storage
C9 Perform a Retrieve operation to obtain Content to verify successful curation

III. Goals of the Demo

To prove the viability of CCFS the demo will implement the model by demonstrating the following:
a. Demo of CCFS exposed as a FUSE (File System in Userspace), both reading and writing files.
b. Demo of CCFS exposed as an http webserver, reading files.
Our prototype of CCFS will use FUSE in order to export CCFS to a Linux operating system as a file system. This implementation will be demonstrated on a laptop with an Intel Core i7 CPU and 8 GB of RAM running Ubuntu 13.10. We will show the time to access files stored in content objects using various combinations of the four object types: Blob, List, Commit, and Tag.

IV. Equipment to be used

The equipment that will be used in the demo includes: 2 Laptops, 1 Consumer home router.

V. Space Needed-Setup Time

The space needed will be approximately a 2 ft. x 3 ft. Table top and we will need 15 minutes to set up.

VI. Additional Facilities Needed

We will need three connections to a power outlet with ground. No wired or wireless connection to any network on Internet will be needed.

VII. Acknowledgements

We thank Priya Bajaj, Mallika Sen, Holly Parrish, and Thomas Saekao of the Opportunity Research Scholars Program within the School of Electrical and Computer Engineering at Georgia Institute of Technology for their contributions to the CCFS code base.