Secure Opportunistic Data Retrievals In Challenged Environments.

M. Chuah, R. Metgzer
Department of Computer Science & Engineering
Lehigh University
chuah@cse.lehigh.edu, rcm2@lehigh.edu

Abstract

Mobile nodes in some challenging network scenarios suffer from intermittent connectivity and frequent partitions e.g. battlefield and disaster recovery scenarios. Disruption Tolerant Network (DTN) technologies are designed to enable nodes in such environments to communicate with one another. Several DTN routing schemes have been proposed. However, very few work has been done on providing information access in such challenging network scenarios. In an earlier work, we studied information retrieval schemes for single-attribute queries in DTNs. Our schemes disseminate replicated data copies and queries to local-neighborhood. To deploy such a system, security issues need to be dealt with. In this paper, we describe a data-centric security solution for opportunistic data retrievals in challenged environments. We first describe our solution, then we describe a preliminary implementation that we have built.

***Index Terms*—disruption tolerant networking, content-based retrievals, data-centric security solution.**

I. INTRODUCTION

With technology advancement, many computing devices e.g. PDAs, smart-phones, sensors have wireless interfaces and hence can form ad hoc networks. Wireless adhoc networks allow nodes to communicate with one another without relying on any fixed infrastructure. These rapidly deployable networks are very useful in several scenarios e.g. battlefield operations, disaster relief centers etc. Routing algorithms designed for adhoc networks, e.g. [1], do not work in challenging network scenarios where node mobility, terrain obstacles, limited radio range, etc often result in frequent network partitions.

Recently, some solutions have been proposed to deal with such challenging communication scenarios. For example, disruption tolerant network (DTN)s [2],[4],[5] are designed to overcome such limitations. Several routing schemes have been proposed for DTNs. Some use history based information to estimate delivery probability of peers and pass the messages to the peer that can best deliver the messages [7],[8]. Such routing schemes rely on the node mobility to deliver packets amidst frequent network partitions using a store and forward approach.

Although routing is an important design issue, the ability to access information rapidly is also an important feature that a DTN should provide since the ultimate goal of having such a network is to allow mobile nodes to access information quickly and efficiently. For example, in a battlefield, soldiers need to access relevant information e.g. terrain descriptions, weather, intelligent information, locations of enemy and friendly forces etc. In [3],[9], we have designed two information retrieval schemes that use query/data duplications to enhance the query success rate in DTN environments. Our results show that the scheme that binary spreads replicated data copies and queries can achieve 45% to 460% higher query success ratio when compared to a scheme that does not use any data and query replication. In [3],[9], we do not address any security design.

In this paper, we describe a prototype that we have built to provide secure opportunistic data retrievals in challenging network environments. Our prototype system uses a data-centric security solution. In our solution, authorized users derive encryption keys for the data items which they are authorized to publish or access from the access keys they get when they authenticate themselves to a mobile key server. Our data-centric solution is scalable since the mobile key server only needs to maintain as many access keys as the number of nodes present in the ontology tree, and the storage nodes only need to store the data items and the subscriptions from users. The rest of the paper is organized as follows: In Section 2, we first give a high level overview of our secure opportunistic data retrieval solution. Then, in Section 3, we provide detailed descriptions of the secure data retrieval system that we have built. In Sections 3.1 to 3.4, we elaborate in details the user authentication procedure, and the operations of the producer, storage node, subscriber respectively. Next, we present in Section 4 a description of a small size prototype that we have created to demonstrate our design. We also discuss the limitations of our current prototype. In Section 5, we discuss related work. Last but not least, we conclude in Section 6 by discussing near future work that we intend to explore

II. OVERVIEW OF SECURE OPPORTUNISTIC DATA RETRIEVALS

In conventional cryptography systems, the security associations are established between the source and

destination pair. For example, SSL creates an encrypted session between a pair of hosts. SSL makes use of both symmetric and asymmetric cryptography. Symmetric key algorithms may be used to encrypt messages sent between a pair of hosts while asymmetric key algorithms are used for key negotiation between these two hosts. When data needs to be disseminated to many hosts, such a pairwise key arrangement is not scalable. Thus, we are interested in exploring a data-centric approach where subscribers of a particular data category are given the same key by either the publisher of the data or a key server. The data-centric approach provides a more scalable solution that is independent of the source/destination pair and the number of subscribers.

A core prerequisite of this approach is of course assigning and maintaining the category information of all published data items in the system. We use an ontology tree based approach for classifying the data while being careful to not predefine the ontology tree so that the system will not be constrained to that particular ontology tree. As the first step, we define a node labeling scheme for a generic ontology tree. All labels are strings, with the root node being a special case with a label of "*". The rest of the nodes within the ontology tree are assigned labels that consist of numbers and underscores. The numbers refer to the numbering of a child at successive levels of the ontology tree. For example, a label such as "2_45_0_235", refers to the 235th child of the 0th child of the 45th child of the 2nd child of the root, assuming the children of a node are labeled from left to right starting at zero. Such a generic labeling scheme can be applied to an ontology tree of any arbitrary structure as long as it conforms to the mathematical definition of a tree. There are two unique and useful features of this labeling scheme: (a) if a node is added to the ontology tree after the system is already running, then that new node will be added as the right most child of the appropriate node and will not change the label of ANY existing ontology node; (b) it is trivial to determine if a node is in the sub-tree of another node by simply performing the longest prefix string matching; if the prefix match includes the entirety of the parent node, then the child node is indeed in the sub-tree of the parent. Each ontology tree is assigned its own ontology tree identifier (treeID) e.g. Google and Microsoft have different ontology tree identifiers. A subscriber may subscribe to data categories offered by different companies (and hence different ontology tree identifiers) so its subscription request must specify both the treeID as well as the node label since a particular node label may mean different things in different ontology trees.

To ensure that published data is only accessed by authorized users, one needs to design a data-centric security mechanism. In our approach, we create a category-based hierarchical key tree such that the encryption keys for the children categories can be derived from a parent but it is hard to derive the parent key given any of the children keys.

III. SECURE DATA RETRIEVAL (SEDAR) SYSTEM

Fig. 1 shows the system architecture of our secure data retrieval (SEDAR) system. In our SEDAR system, there are 4 types of network entities namely (a) publishers, (b) subscribers, (c) storage nodes, and (d) a mobile key server (MKS). Each node has a unique endpoint identifier (EID).

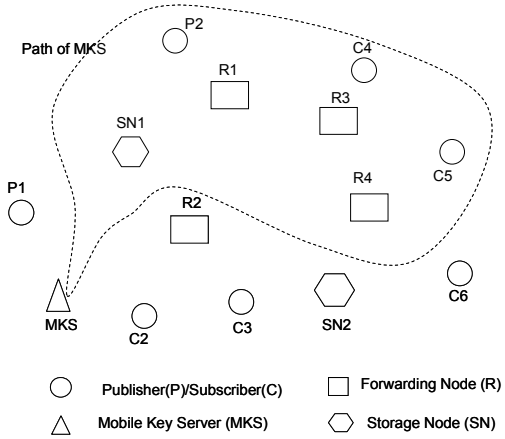


Fig. 1: Secure Opportunistic Data Retrieval System

Data items generated by the publishers are encrypted using AES. A plaintext meta-data describing the data item is added to the encrypted data block. The encrypted data item together with its plaintext meta-data descriptor is then sent to a storage node. Subscribers send their subscriptions to a particular storage node. The storage node will send all currently stored data items or those which are subsequently received that match the subscription to any subscriber. A subscriber which subscribes to a particular ontology node implicitly subscribes to all data items that belong to the categories which are children of that ontology node.

The keys used for encryption and decryption follow a very specific scheme themselves. A system-wide key for the root node of the ontology tree is stored at the Mobile Key Servers (MKS). The keys for the other data categories (represented by the ontology tree nodes) are derived from this root key. The key derivation scheme is designed such that given the key for any particular node within the ontology tree, you can derive the key for any of its children, but you CANNOT derive the key of its parent. To derive the key of the child node, we take the key of the current node and append the number of the child node to it. Then, the augmented key is hashed via Secure Hash Algorithm (SHA), the resulting hash is the key of the child. For example the key of 3_45_6 is SHA(SHA(SHA(root key||3)||45)||3), where || is the concatenation operator.

Let us consider the hierarchical key tree shown in Fig. 2. The key tree is designed such that given a parent key, all the child keys can be easily derived but given a child key, it is computationally infeasible to derive a parent key. For example, in Fig 2, the encryption keys must be designed such that a subscriber that wants to access data items that belong to "man-made" category, and another subscriber that wants to access data items in the "terrain" category can derive the encryption keys used to encrypt all data items in the "tunnels"

subcategory. The subscriber that only subscribes to data items in the "weather" category should not be able to derive encryption keys for any data items in the "tunnels" subcategory.

We describe how an encryption key *Ke* for a data item, and an access key *Kf* for a subscription filter *f* are derived. Let *kls* be a string that represents a subcategory. For example the *kls* for the "man-made" subcategory in Fig 2 is 2_2. The encryption key for any data item within the category c is denoted as:

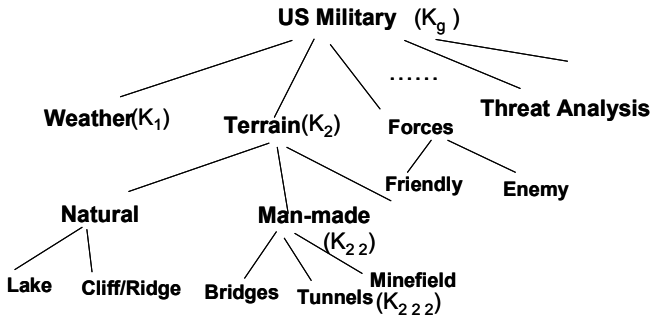
$$Ke=K_{kls}\ (c)$$

Let us assume that the root element for Publisher *P1* uses the key K_g for the "US-Military" tree. To determine the encryption key, *Ke*, for the data items in the "tunnels" category, a publisher first determines the string, *kls*, that represents the "tunnels" category, which is 2 2 2. Then, $Ke=H(H(H(K_g \| 2)\|2)\|2)$.

A subscriber can subscribe to any category within the category-based hierarchical tree. Assume that a subscriber subscribes to the "terrain" category, then he has the access key, K_2. When he wants to access the data items belonging to the "tunnels" category, he knows that the *kls* for that category is 2 2 2 so he needs to perform the following operation to his access key to derive $K_{tunnels}$, the encryption key for data items that belong to the "tunnels" subcategory.

$$K_{tunnels}=H(H(K_2 \| 2)\|2).$$

Note that a subscriber is allowed to subscribe to multiple categories. For example, a subscriber may subscribe to items related to both the "weather", and "man-made" categories. This subscriber will be given the access key for the weather category i.e. K_1, and the access key for the man-made category i.e. $K_{2\,2}$.



$$\ldots\ldots K_{Weather}=H(K_g \| 1)$$

$$K_{Man\text{-}made}=H(K_2 \| 2)$$

Fig. 2: Category-based hierarchical key tree

In our data-centric security solution, storage nodes do not have the keys to decrypt the encrypted data items. Thus, even if a storage node is compromised, the enemy cannot decrypt the encrypted data items. To prevent a malicious node from pushing fake published data items to a storage node, a publisher may attach the token it receives from the MKS during the user authentication process to every data item that the publisher wishes to push to the storage node. Similarly, a subscriber is issued a token when it authenticates itself with the MKS. This token must be attached to any subscription request message that a subscriber sends to a storage node. The token may be a digital signature created using the mobile key server's private key so that the storage node can verify that a publisher (or subscriber) has authenticated itself with the MKS. The token can be refreshed periodically (which means the publisher needs to re-authenticate itself with the publisher periodically) so that the possible damage caused by a compromised publisher can be minimized. The storage node can also set a maximum limit to the number of new data items which a publisher can publish or the maximum number of data items which a subscriber can retrieve.

In the event, a publishing or subscribing node is compromised, only the keys for the categories they are authorized to publish or retrieve will be compromised. Upon detecting such an event, the mobile key server can send a revocation message to all storage nodes to prevent the compromised nodes from publishing new data items or retrieving existing data items. The MKS will also issue new tokens to existing publishers, subscribers, and storage nodes.

In subsequent sections, we elaborate on (a) the authentication procedure performed by a user with the Mobile Key Server to retrieve encryption keys that are used for encrypting published data items or for decrypting the subscribed data items that have been retrieved, (b) the publishing procedure, (c) the storage node procedure, and (c) the subscribing procedure.

A. Authentication:

As a security system, it would of course be completely useless without some method for constraining who has access to what. There are two components for the authentication function: namely a Mobile Key Server (MKS) that holds the root key and a user authentication client. You must authenticate in order to run either a publisher or a subscriber application since both applications require the use of encryption/decryption keys for handling the data. When a user wishes to run either one of these applications at a DTN node, then he/she runs the user authentication application. The user authentication (UA) application is configured with a list of MKSs to authenticate against since one needs to authenticate with one MKS in each system which one wishes to publish or subscribe to. The UA application will prompt for a username and password for each MKS in the list. It will create a messate that contains both the entered username and password, then encrypt the message using the Identity-based Encryption (IBE) approach, and send it to the corresponding MKS. The MKS stores the usernames and the SHA hash of their passwords in a file. Thus, if the password file is ever compromised, then the passwords themselves are not compromised. When the MKS receives an authentication request packet, it decrypts the message, and extracts the username and password. Then, it hashes the password, and

compares it with the hash value stored on file for that username. If either the username doesn't exist or the hash does not match, then an authentication failure message is returned. Otherwise, a response message that contains the ontology tree ID and a list of keys for the data categories that this user subscribes to is constructed. The list of keys contains a list of tuples, each tuple is an ontology nodeID and its associated key. The included ontology nodeIDs represent all the data categories that this user is authorized to have access to. This return message is then IBE encrypted and returned to the client. The client then takes this message and stores it in a file after decrypting it. The filename that stores the returned keys is derived from the EID of the MKS that the client contacts. If the client contacts and authenticates with multiple MKSs, then multiple files will be created. Fig 3 illustrates the authentication process and an example of the contents of the access key file.

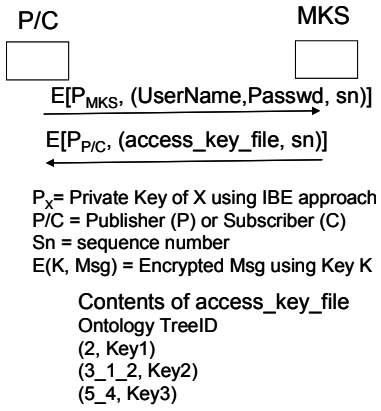


Fig. 3: the user authentication process

The producer and subscriber, as part of their configuration, are told the EID of the MKS to authenticate against. As mentioned earlier, the actual authentication is handled by the client authentication program. Here, the producer or subscriber application uses the EID to derive the filename that it needs to access to obatin the ontology tree ID, and the access keys of all data categories that the user is allowed to publish or retrieve.

B. Producer:

Before a user is allowed to publish data items, he/she needs to authenticate himself/herself with a MKS node. The EID of the MKS is configured at each DTN node. A user invokes the authentication process described in Section 3.1. After that, the DTN node will receive from the MKS a file that contains the ontology treeID, a token from the MKS, and all the keys associated with data categories that this user is allowed to publish. For example, a user may be allowed to publish data items in the "weather" and "man-made categories" so the file will contains keys for those two specified categories.

A DTN node can run multiple instances of the producer application. Each producer instance will generate data items that belong to one data category. Each producer instance is identified using a concatenation of the process identifier (PID) and the application EID e.g. monkeywrench_dc_center_PID.

When a producer application is invoked, a configuration file is given to the program. This configuration file contains the following information: (a) the EID of the MKS, (b) a watch directory where the program will monitor for new data items that need to be published, (c) the associated ontology node label (e.g. "man-made" category), (d) the published ontology node label (e.g. "bridges" sub-category within the "man-made" category), and (e) a list of storage nodes to which the data items will be pushed.

A producer needs to derive encryption keys that are used to encrypt the data items to be published. The producer first extracts the key of the associated ontology node label (e.g. the "man-made" category) and then derives the key for the published ontology node label (e.g. the "bridges" subcategory).

At startup, a producer will publish all the data items that are found in that specified watch directory. After that, the directory is monitored periodically to see if there is any new or updated data item. If there is, then the producer will push a copy of the new or updated data item to each of the storage nodes that are listed in the configuration file.

Each data item to be published is first encrypted via AES with the appropriate key and a random ephemeral initialization vector (IV). The random ephemeral IV is appended in plaintext to the beginning of the encrypted data item. We refer to this as the encrypted data object. A meta-data for the encrypted data object is appended at the beginning. The meta-data contains three pieces of information, namely (a) the ontology tree ID that was received from the MKS, (b) the ontology node label that this producer is configured with, (c) an object identifier. In our implementation, this object identifier is a file identifier. This file identifier is constructed by concatenating the application EID, the PID of the producer instance, and the original filename of the data item that needs to be published, but all special characters like '/' and ':' are changed to a '_'.

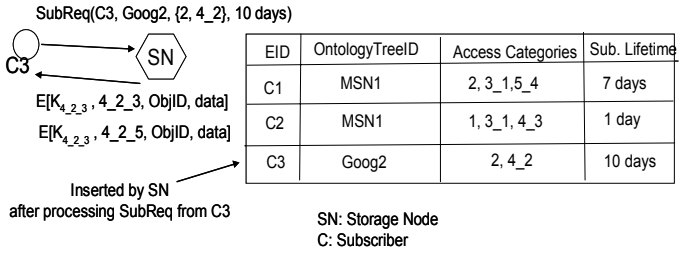
When an object gets updated, and the updated object is published, the desired behavior is that the new version replaces the old version in the storage nodes and subsequently in the subscriber nodes. Since the object identifier is a filename in our implementation, when a storage or subscriber node receives a file with the same file identifier it already has, then it knows that this is an updated version and acts accordingly.

C. Storage:

The storage nodes are used to store all the published data items. All data items that are sent to a storage node by a publisher are accepted by the storage node. When the storage node receives a message from a producer, the meta-data is stripped off the message, and stored in an internal array and the encrypted data item is stored in a special directory at the

storage node. The data item descriptor is the ID contained in the meta-data of that data item. In our current implementation, this identifier is a file descriptor.

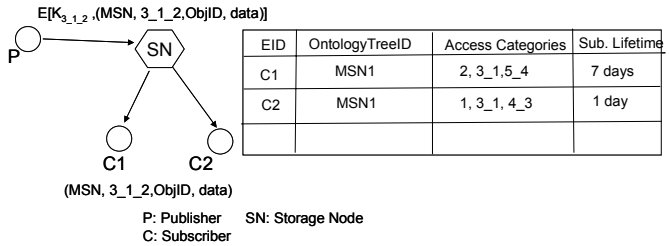
When the storage node receives a subscription request, the subscription request contains (a) an ontology tree ID, and (b) an ontology node label. The storage node stores this information in an array. Then, the storage node scans the meta-data list it maintains for all stored data items, and looks for any entries that have the same ontology tree ID and ontology node that is within the sub-tree rooted at the ontology node specified in the subscription request message. If there is a match, then the file is sent along with its associated meta-data. Figure 4 shows the actions performed by a storage node upon receiving a new subscription from the subscriber C3. The storage node inserts the new subscription information into the subscription table that it maintains. Then, the storage node discovers that there are two data items in its storage that match the subscription. Thus, these two items are sent as separate bundles to the subscriber C3.



EID	OntologyTreeID	Access Categories	Sub. Lifetime
C1	MSN1	2, 3_1,5_4	7 days
C2	MSN1	1, 3_1, 4_3	1 day
C3	Goog2	2, 4_2	10 days

Fig. 4: Actions taken by Storage Node upon receiving a new subscription

When the storage node receives a new data item, the storage node compares the meta-data descriptor of this new data item with the subscription list it maintains to see if there is any match. If there is a match, a copy of that new data item will be sent to the subscribing node after the new data item and its meta-data is stored. In Figure 5, we illustrate how a storage node takes actions upon receiving a new encrypted data item from a publisher. The storage node first stores the data item that belongs to the category 3_1_2. Then, the storage node checks the subscription table it maintains and discovers that this new encrypted data item needs to be sent to subscribers C1 and C2. Subscribers C1 and C2 can derive the encryption key for the category 3_1_2 from the access key they possess for the category 3_1. Thus, both of them are able to decrypt the encrypted data item.



EID	OntologyTreeID	Access Categories	Sub. Lifetime
C1	MSN1	2, 3_1,5_4	7 days
C2	MSN1	1, 3_1, 4_3	1 day

Fig. 5: Storage Node pushing new data items to existing subscribers.

D. Subscriber:

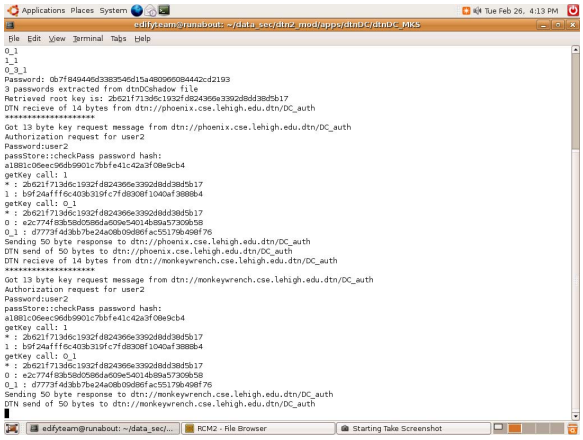
As in the producer case, a user needs to authenticate himself/herself with the mobile key server before he/she is allowed to run a subscriber application at a DTN node. We assume that the mobile key server contains information about the data categories that a user is allowed to access. After the authentication process, the DTN node at which a user runs the authentication application will have a file which stores the ontology treeID, and all the access keys for the data categories that this user is allowed to access. The user can then run the subscriber application for accessing data items of any data category that this user is allowed to access. Multiple instances of the subscribe application can be run: each instance will retrieve data items of a particular category that the user is authorized to access.

Using the example in Fig 2, let say the subscriber is allowed to access the "weather" category and all categories that are under the "man-made" category. The key file contains the access keys for the "weather", and "man-made" categories. These keys are referred to as the association node keys. A user can then request data items in the "bridges" category. To accomplish this, a subscriber application will be invoked with a configuration file. The configuration file contains (a) the EID of the storage node from where the data items will be retrieved, (b) the application EID of the MKS, (c) the ontology treeID, (d) the access node label (e.g. "man-made" category in our example), (e) the data category that this subscriber wants to retrieve (e.g. "bridges" in our example) and (f) the storage directory where the decrypted data objects (implemented as files in our prototype) will be stored. The subscription request message is sent only to the configured storage node.

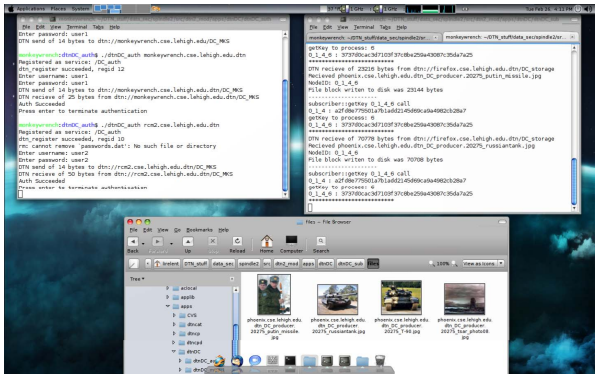
The storage node will send each encrypted data item along with its meta-data in a bundle. When the subscriber receives the bundle, it extracts the encrypted data item and its meta-data. Then, it looks at the ontology node label within the meta-data, and derives the appropriate key from the association ontology node key that it obtains during the authentication process. It then uses the derived key to decrypt the file block. For example, using the key tree in Fig 2, if a subscriber retrieves data items in the "bridges" category (with label 2_2_1), and has the key for the "man-made" category (with label 2_2), then, the subscriber derives the encryption key for the "bridges" category as SHA(key("man-made")‖1). Using the object identifier (which is a file descriptor in our prototype system) contained in the meta-data as the filename, the subscriber creates a file in the storage directory. Then, it dumps the decrypted data into that file. By using the unique object ID as the filename, we can prevent data items with the same name from different producers from overwriting one another. In addition, our approach allows us to update an existing file with its new version.

IV. TESTBED DEMONSTRATION

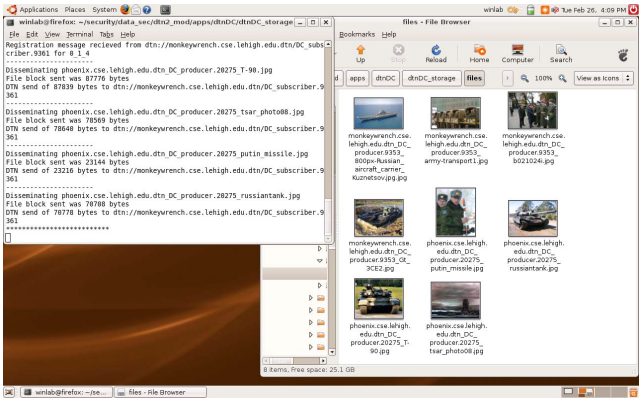
We have built a preliminary prototype. Our prototype is written in C++ and runs on laptops which run Ubuntu and DTN2 reference code. There are several important modules in our prototype: namely a publisher, a client, a storage, and an authentication & key distribution module. A four-node testbed is built. One node, *rcm2*, acts as the mobile key server (MKS) with whom any publisher or subscriber can authenticate themselves and obtain the authorized access keys for the data categories they are authorized to access or publish. A second node, *firefox*, acts as a storage node which stores all the data items that publishers send. A third node, *monkeywrench*, and a fourth node, *phoenix*, each runs both a publisher and a subscriber process. A node needs to authenticate itself using a password with the mobile key server. After it is authenticated, it will get a file containing access keys for all the authorized data categories. Figure 6(a) shows the outputs at the MKS where the 2 nodes obtain their access key files. Figure 6(b) shows the data items retrieved by the subscriber running on the node *phoenix*. Figure 6(c) shows the data items stored at the storage node. These data items are sent to the storage node by the two publishers running on the nodes, *monkeywrench* and *phoenix*.



(a) output from the mobile key server



(b) output from the subscribing node



(c) Output from the storage node

Figure 6: Screendumps from different nodes

Figure 7 shows the encryption/decryption time for different data item sizes. The experiment was conducted on a laptop running our data centric security solution. The laptop runs Ubuntu and has a 2.0GHz Intel Pentium M processor with 1GBytes RAM, 2 Mbytes CacheSize. The results indicate that the encryption and decryption times using our approach is reasonable.

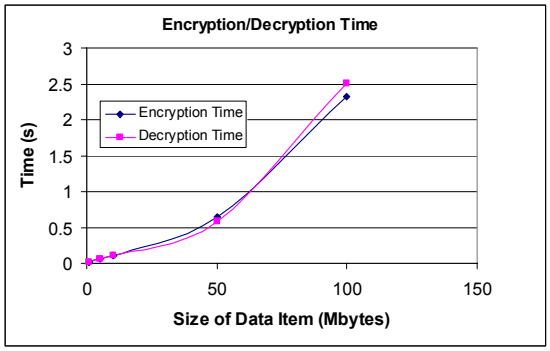


Fig. 7 Encryption/Decryption Time of Objects with Different Sizes

A. Limitations of Current Prototype

Our current system uses files to represent data objects. Our prototype is also not integrated yet with any semantic ontology framework. In the near future, we intend to create and maintain a database for all the stored objects so that semantic related indexes can be built. Semantic related indexes allow data objects to be searched more easily. We also intend to integrate our prototype with a semantic ontology tree framework so that a larger scale experiment can be conducted.

Our current prototype does not allow the user to dynamically negotiate with the mobile key server for adding new categories into its allowable publish or access list. This feature will be useful in many applications e.g. a user may only subscribe to all intelligent information related to Baghdad initially but may be interested in retrieving sensor data related to nearby cities as his convoy moves outside of Baghdad. The subscriber may also choose to terminate previous subscriptions.

V. Related Work

DTN Routing Schemes

DTNs have been proposed to deal with challenging communication network scenarios e.g. networks with intermittent connectivity, large delay and extended network partitions. DTNs can be useful in many scenarios e.g. vehicular scenarios [8], wild life monitoring [10], deep space communications [2] and military networks [12]. In [4],[5], an architecture based on store-and-forward concept has been proposed. A summary of the different DTN routing approaches can be found in [14].

Secure Data Retrieval

In [15], the authors describe a privacy enhanced data-centric sensor network that allows sensor data with the same "name" to be stored in the same location so that queries for data of a particular name can be sent directly to the storing nodes using geographical routing protocols. In their design, a certain geographical area is divided into different cells. The nodes within a cell *u* determine the location of the storage cell *v* through a keyed hash function. Cell u encrypts the sensor data with its cell key and forwards the encrypted data item towards the storage cell, *v*. Upon receiving the message, *v* stores it locally. An authorized mobile sink (MS) interested in the data item can determine the storage cell *v*, retrieves and decrypts it with the proper cell key. The approach in [15] is only suitable for static sensor nodes. For mobile sensor networks with intermittent connectivity, the approach in [15] performs poorly especially with sensor data which has short expiration time.

In [16], the authors propose a bucketing scheme to store the data items with attributes values within certain ranges, and employ encoding numbers to prevent storage nodes from dropping data. In [16], the authors assume that each publisher performs the clustering to determine the bucket size and sends the encrypted data to the nearest storage node. This means that range queries need to be flooded to all storage nodes. In addition, each publisher encrypts all the data using the same key. Our approach differs from [16]: (a) we let the storage nodes to do the clustering, (b) we allow storage nodes to be selected based on encounter opportunities rather than being pre-selected, (c) we suggest data encryption keys that are based on subscriptions or attribute range, not on a per publisher basis. By letting the storage nodes do the clustering, the false positive (defined by the authors in [16] as the redundant data included in the response due to the fact that a publisher encrypts all data items within a certain range together) in our system will be smaller.

Content-Based Network

The authors in [17],[18] propose a content-based network where the nodes run two routing protocols: a broadcast and a content-based routing protocol. Their content-based routing protocols provide both push and pull mechanisms. The advertisements from the receivers are pushed to all potential senders. Each node that receives such advertisements will check if it already supports a content-based address (a predictor) that covers the new advertisement. If so, it simply drops the new advertisement. If not, it computes the set of next-hop links on a global broadcast tree rooted at that receiver and forwards the receiver's advertisement along those links. The intermediate node also updates its content-based routing table. A pull mechanism is also provided in their design to allow nodes to send a send-request message to pull the latest selected predicates from all nodes. However, the authors do not address any security design.

VI. Concluding Remarks

In this paper, we have presented a prototype system for secure opportunistic data retrievals that work in challenging network environments. Our system uses a data-centric security solution where publishers or subscribers authentication themselves with a mobile key server to obtain access keys that allow them to publish encrypted data items or decrypt encrypted data items that are retrieved from the storage nodes. The storage nodes do not have the relevant keys to decrypt the data items. Thus, even if the storage nodes are compromised, the data items remain secure. There are several limitations in our current prototype system. In the near future, we intend to add new features to our prototype system to remove these limitations.

Acknowledgment

This work has been supported by DARPA under Contract W15P7T-06-C-P430. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsor of this work.

References

[1] S. Das, C. Perkins, E. Royer, "Performance Comparison of Two On-demand Routing Protocols", Proceedings of IEEE Infocom, March 2000.

[2] S. Burleigh et al, "Delay Tolerant Networking – an approach to interplanetary internet", IEEE Communication Magazine, June, 2003

[3] M. Chuah, P. Yang, "Performance evaluation of data-centric information retrieval schemes for DTNs", Proceedings of Milcom, Oct, 2007.

[4] K. Fall, "A delay-tolerant network architecture for challenged Internets", *Proceedings of SIGCOMM*'03, August 2003.

[5] V. Cerf et al, "Delay-Tolerant Network Architecture", Internet Draft, draft-irtf-dtnrg-arch-02.txt, July 2004.

[6] D. Eastlake, P. Jones, "US Secure Hash Algorithm 1(SHA1)", RFC3174, Sept 2001.

[7] A. Lindgren, A. Doria, O. Schelen, "Probabilistic routing in intermittently connected networks", Sigmobile, Mobile Computing and Communications Review, Vol 7(3), pp 19-20, 2003.

[8] J. Burgess, B. Gallagher, D. Jensen, B.N. Levine, "MaxProp: Routing for Vehicle-Based Disruption-

Tolerant Networking", to appear in Proceedings of IEEE Infocom,April,2006.

[9] M. Chuah, P. Yang, "Data-Centric Information Retrieval Schemes for DTNs", CSE Technical Report, March, 2008.

[10] P. Juang, etc, "Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet", Proc ASPL0S, Oct 2002.

[11] H. Krawczyk, M. Bellare, R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", Feb, 1997.

[12] DARPA DisruptionToleerant Networking Porgram, http://www.darpa.mil/ato/solicit/DTN, July, 2006

[13] R. Rivest, "The MD5 Message Digest Algorithm", RFC1321, April, 1992

[14] Z. Zhang, Q. Zhang, "Delay/Disruption Tolerant Mobile Ad Hoc Networks: Latest Developments", WCMC, Vol 7, Issue 10, pp 1219-1232.

[15] M. Shao et al, "pDCS: security and privacy support for data-centric sensor networks", Proceedings of Infocom, 2007.

[16] B. Sheng, Q. Li, "Verifiable Privacy-Preserving Range Query in Two-tiered Sensor Networks", to appear at IEEE Infocom, 2008.

[17] A. Carzaniga et al, "A routing scheme for content-based networking", Proceedings of Infocom 2004.

[18] A. Carzaniga and A. L. Wolf, "Content-based networking: a new communication infrastructure", NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Scottsdale, Arizona, Oct 2001.

[19] A. R. Bharambe et al, "Mercury: supporting scalable multi-attribute range queries", Proceedings of ACM Sigcomm, Aug/Sept, 2004.

[20] F. Hess, "Efficient Identity based Signature Schemes based on Pairings", Proceedings of SAC 2002, LNCS 2595 p310-324, St Johns, Newfoundland, Springer-Verlag, 2003.

[21] M. Chuah, J. Han, "Performance studies of information retrieval schemes for multi-attribute queries in DTNs", CSE Technical Report, March, 2008.

[22] T. Harder et al, "Node labeling schemes for dynamic XML documents reconsidered", Data & Knowledge Engineering, Vol 60, Issue 1, pp126-149, Jan, 2007.