Multi-Factor Authentication for More Resilient Distributed Storage in Wireless Networks
[Technical Report]

Scott Bell
Computing and Information Sciences
Kansas State University
Manhattan, KS 66506
Email: rsbell@ksu.edu

Eugene Vasserman
Computing and Information Sciences
Kansas State University
Manhattan, KS 66506
Email: eyv@ksu.edu

Daniel Andresen
Computing and Information Sciences
Kansas State University
Manhattan, KS 66506
Email: dan@ksu.edu

***Abstract*—Modern military units derive great tactical advantage from secure real-time sharing of data such as maps, images and orders among soldiers. However, distributing this information in real time in a combat situation creates significant risk: when using a mobile communication network, the adversary may capture one or more mobile devices, gaining access to this data and endangering the entire unit. While these devices are generally tamper-resistant and require a login, this would not deter a well-funded and motivated attacker. In this work, we present a protocol which significantly reduces the adversary's window of opportunity for attack by incorporating distributed content storage and revocable authentication for users and individual devices *without increasing the difficulty of soldier-device interaction*. To further limit an adversary's ability to access this data, file requests must contain fresh authentication information from both a trusted user and device. We analyze the benefits and trade-offs of this protocol both theoretically and through tests on real-world mobile devices, and find that the computation, response, and battery overhead are acceptable purely in software, and can be greatly reduced with inexpensive hardware acceleration. Simulations indicate the probability of data loss is significantly reduced over systems which only require a user secret (password or PIN).**

I. Introduction

Today's military units employ rapid movement and tactics that require real-time secure communication channels. The ability for soldiers to access files such as maps and satellite images provides a huge tactical advantage. The effectiveness of this technology is evident in the Blue Force Tracker system currently used in military vehicles to display real-time locations of friendly (and enemy) forces and tactical information [11]. While this technology can be installed within a vehicle, the battery capacity and antenna requirements necessary for this real-time satellite communication prevent the deployment of such a system to the individual soldiers. One of the most versatile methods for disseminating and displaying information in real time, and the current tool of choice, is a network of handheld wireless devices incorporating radio communication, data storage and display capability [22].

However, the convenience provided by these devices leads directly to vulnerability — because they are small and lightweight, they can be easily lost or stolen, potentially giving adversaries access to all the information the owner would have been able to access. To prevent an adversary who captures a device from immediately gaining access to all stored data, devices encrypt stored data and require users to enter a short PIN. The length and complexity of PIN codes are unfortunately limited by the requirement that PINs be entered quickly, making them vulnerable to brute-force cracking once a device is captured, especially by a nation-state adversary with the associated resources and computing power. Although military-use mobile devices frequently incorporate tamper-resistant technologies, these will not deter nation-state adversaries in the long term. Therefore, we must assume that the adversary will eventually gain access to the decrypted content of the device local storage.

An alternative approach to data protection is to encrypt then fragment files, distributing file fragments across multiple devices. When capturing a device, the adversary would only gain access to fragments of encrypted files — which are useless without obtaining additional file fragments from other devices and the decryption key. Using this strategy of as-needed retrieval and peer-to-peer (P2P) storage that distributes fragments across multiple devices for failure-tolerant retrieval [10], [26], we avoid the single device capture vulnerability. However, this introduces an even greater problem — if a captured device's network access is not revoked quickly, the adversary may now access *all files in the network* given a single compromised device as a "gateway." Prior work has not addressed the use of "active" but compromised devices, assuming either that tamper resistance or PINs are sufficient protection, or that devices are never lost. Neither assumption is safe given a well-funded and skilled adversary in a military context.

In this work, we introduce, analyze, and implement a new protocol for practical secure storage and access of data on mobile devices in a battlefield, resilient to device capture by a powerful nation-state adversary, *without relying on strong hardware tamper resistance*. Our system does this without sacrificing usability in the field or incurring prohibitive computation, time, or battery consumption costs.

We eliminate the assumption that any device (software or hardware) which connects to the network be trusted to use the on-demand file access, and greatly reduce the window of vulnerability during which captured devices may act as

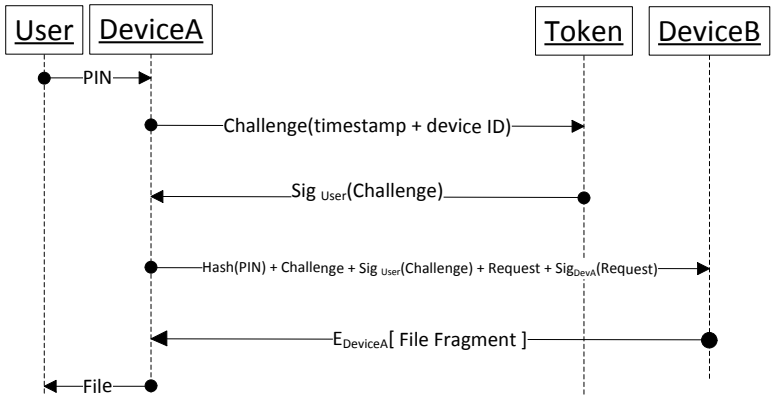


Fig. 1: Message passing protocol for mobile file access. Note that inter-device communication is assumed to be secure.

"network gateways." Further, we allow devices and users to be revoked independently, removing the requirement that a user must be paired to a single device. *Each device within the network verifies that every file fragment request to which it responds comes from a non-revoked device, a non-revoked user, and was issued recently (preventing replay).* As files are reconstructed dynamically from the network, and never stored, we no longer need to rely on device tamper-resistance technologies.

Latency is a concern when using real-time file retrieval, but it can be greatly reduced using erasure coding for file storage and threshold cryptography to protect keys [10], [26]. For instance, a file is split into n fragments which are distributed across n different devices, but only $m < n$ of these fragments must be retrieved in order to reconstruct the file. This has two benefits: 1) latency is minimized by requesting fragments in parallel, with more than m requests in one batch, accounting for the inevitable lack of response from some devices; and 2) security is greatly enhanced, since an adversary controlling fewer than m devices cannot learn any information about the stored contents.

The presence of a user in our system is verified by a challenge-response from a user-specific wearable token. Without a response from a valid token, devices will not unlock, will not make network requests, and other devices within the network will not respond to file fragment requests the adversary may attempt to make. We extend this idea to individual mobile devices as well — a currently trusted device must transmit user file fragment requests, and other devices will ignore requests from a device which is suspected as being compromised. Our system follows the protocol for personal identification verification (PIV) as specified in FIPS 201-1 [20] for authentication using asymmetric cryptography in a networked system.

We analyze the benefits of this system in two ways. Using an attack tree, we show the probability that an adversary may access the network using a compromised lost or stolen device essentially reduces to the requirement of capturing a legitimate user along with his or her device. In addition to analyzing the benefits of our design theoretically, we developed a prototype implementation and demonstrate experimentally that the costs of these security mechanisms (in terms of increased response time and battery consumption) are acceptable with a software-only implementation, and can be further reduced through inexpensive specialized hardware [7], [15].

The rest of this paper is organized as follows: Section II describes the system design; Section III lays out the details of our prototype implementation and gives the results of evaluations; Section IV discusses related work; and finally Section V presents a summary and future work.

II. Design

Our design focuses on reducing the threat posed by an adversary capturing a limited number of devices, and consists of three primary components: the mobile devices participating in the wireless network, the users, and user-wearable tokens. Each token is bound to a specific user while mobile devices are not. This allows a user to log into any device and gain access to the distributed data files — like radios or physical maps, devices are interchangeable. When logging into a device, a user must enter a short numeric PIN associated with that user. The PIN must be simple in order for the device to be useful in combat, so we do not assume that this PIN provides strong protection against a skilled adversary, and treat it more like a user identifier. The device uses this PIN to access both connection information for that user's wireless token and the user's public key. The device then attempts to strengthen user authentication by locating the token, which contains a much stronger cryptographic key. The PIN in combination with the presence of the correct token confirms the user's identity both to the local device and to other devices on the network when making requests for file fragments. Figure 1 shows the communication process which occurs between the system components.

A. Token

The token is a self-contained, tamper-resistant wireless device containing a user's unique private key, corresponding to the public key known by all mobile devices within the network, and associated with the user's unique PIN (this is explained more thoroughly in Section II-B1). The token provides a cryptographically secure means of authenticating a user without

increasing the user's involvement in the authentication process. The token and a given device are initially paired to one another by the user initiating the pairing sequence on the device and pressing a button on the token to verify the pairing operation. Once paired in this manner, the token ignores connection requests and challenges from other devices until it is paired to a different device using this same procedure. Once a mobile device connects to the token, the device can issue challenge messages which the token signs using the user's private key and then returns to the device.

In operation, a token can take many forms and utilize one of several different wireless technologies. We envision a ubiquitous wearable device such as a watch, ring, dog tag, or smart card, whose loss is conspicuous to the user. This makes it easier for users to keep the token on their person at all times and also makes it more difficult for an adversary to acquire a token without a user's knowledge, as opposed to a mobile device which may be dropped or forgotten. The token could use wireless communication such as standard Bluetooth, Bluetooth LE [2], ZigBee [27], Near Field Communication [18], RFID [23] or any other short range wireless protocol; the choice of technology is based on application-specific requirements such as cost, operating environment, token concealment, durability, availability, and battery life expectations. The transmission range for token communication should be limited to a few feet to reduce battery consumption and to help ensure that the soldier who is in possession of a specific token is the same person using the mobile device. The pairing procedure also limits an adversary's ability to connect to the token remotely since the pairing process must be verified by the user pressing the pairing button on the token.

B. Device

Each mobile device provides a conduit for a user to access files stored within the mobile network, and responds to requests from other devices for file fragments which it possesses. Like each user, each device has a device-specific key pair. The state diagram in Figure 2 shows the progression of states for a device once it has been initialized. Each state will be discussed in order.

1) Initialization and Startup: All devices participating in the system are initialized at the start of a mission with the following information:

- The device's own ID and key pair
- The ID and public keys for all other devices
- A hash of the unique PIN assigned to each authorized user along with:
    - Each user's public key
    - Each user's token connection details
- An administrative account public key
- Loose clock synchronization information
- File fragment IDs and where each is stored
- The file fragments the device itself should store

The list of hashed PIN values and public keys allows each device to associate a user who enters a given PIN with the correct public key that is used to authenticate the signed challenge received from the user's token (see Figure 1). The list of other device IDs and public keys is used to authenticate file fragment requests received from other devices. The administrative key is used to authenticate commands from an account which will be used only to revoke devices or users or to issue remote disable commands. Device clocks are loosely synchronized so that each device can accurately determine the freshness of requests made by other devices. No file is stored entirely on any single device, only fragments of the encrypted files are stored on devices. Once initialized, a device starts in a locked state where no file requests are allowed to be sent out.

2) User Authentication: As Figure 2 shows, before granting a user local access, a device first requires a PIN from the user. Longer PINs are more secure but can be more difficult to remember as well as more difficult to enter when under stress. Therefore, we utilize the token to reduce the need for prohibitively long PINs, and treat the PIN more like a user ID than as a security component when analyzing the security of our system. Once a user enters a valid PIN, the device locates the token associated with a hash of that PIN, and if necessary, attempts to pair with the token (if they were not previously paired as described in Section II-A). Then, the device issues a challenge, which the token signs and returns. This challenge contains a timestamp to prove the freshness of the message, and includes the ID of the mobile device to show which device issued the challenge. The device verifies the signature received from the token using the public key that matches the hashed value of the PIN, validating the user-PIN-token pairing. At this point, the device has two-factor authentication of the user and the user is allowed to request files within the system. If the signature cannot be verified the device will drop the connection to the token and return to the initial locked state. *Note that devices never store PINs "in the clear," so PINs cannot be recovered from captured devices. Even if the hash of a PIN were recovered, it is useless without a valid challenge signed using the corresponding private key contained on a user's token.*

Stronger mechanisms for initial user identification are available such as biometric scans (e.g., fingerprint or iris). Implementing one of these mechanisms in place of using a PIN would increase the time it takes an adversary to break into the mobile device, thus making it more likely that the device is discovered as missing before the adversary can access the file system. In our analysis, we assume that the PIN does not provide protection for the device and treat it as being cracked immediately upon device capture. Incorporating any of these stronger mechanisms for user identification would result in even greater improvements in system security.

3) Logout/Timeout: The device periodically sends fresh challenges to the token to ensure that the token is still within transmission range. Each response is verified and replaces the previous challenge response. If the token is not detected or responds incorrectly, the device returns to the "locked" state (Figure 2) and any files that are currently open are securely deleted from memory. From the locked state, the user is required to reenter a PIN and the token must be detected in order to regain access to the device. Additionally, if the device does not detect user interaction over a specified period of time (the device is idle), it logs the user out and returns to the "locked" state. Both of these features limit the window of opportunity for an adversary to recover a device which is in a

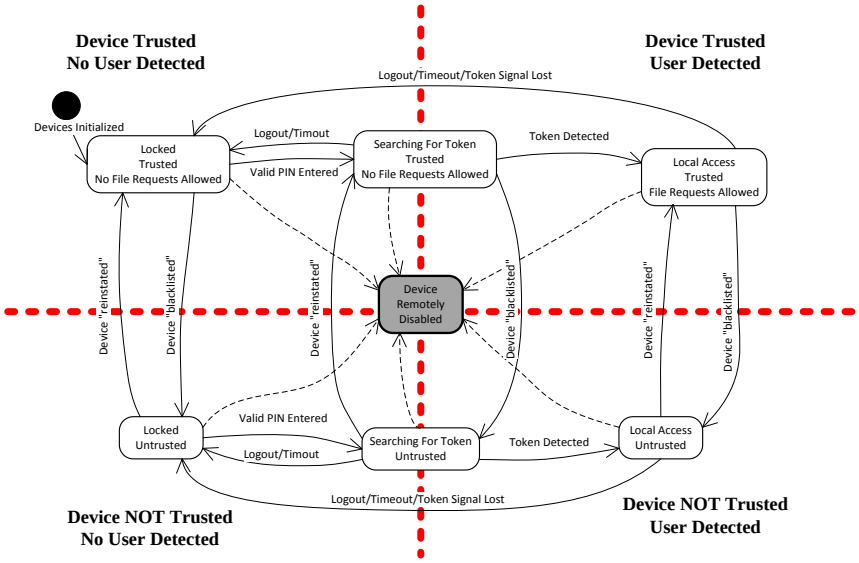


Fig. 2: State diagram for the mobile device, including transitions. Note the four different "trust quadrants."

state that allows access to files in the system.

4) File Requests: As indicated in Figure 1, the next step is the requesting device sending requests for file fragments to other devices within the mobile wireless network. Each request must contain enough information to prove the following:

- A currently authorized user is in possession of the device sending the request
- The user authentication information is fresh and was obtained by the requesting device, and
- The requesting device is currently authorized.

When a user requests a file, the device uses its local information to determine which other devices in the network hold fragments of that file, and constructs requests for $m < n$ fragments, where m is sufficient to reconstruct the file locally. If fewer than m requests succeed, the device can send additional requests to other devices until it receives at least m fragments. The initial number of requests can be increased to $m + \epsilon$ in anticipation of some devices failing to respond. The details of routing within the network is beyond the scope of this paper, but it is assumed that communication between pairs of devices is secure, preventing an adversary from eavesdropping on messages sent across the network.

Individual file fragment requests provide multi-factor authentication by including authentication information for both the user and the requesting device, along with the identity of the requested fragment. Specifically, the token challenge response and a hash of the user PIN authenticate the user, while a signature of the request using the requesting device's secret key authenticates that device. *All of this is required, along with freshness information, before other devices will honor a request.*

The responding device verifies this information using locally stored data. It also checks the freshness of the timestamp contained in the challenge, and verifies that the device ID in the challenge matches the ID of the requesting device. If all of these checks are acceptable, the responding device returns the file fragment to the requesting device. If not, the request is ignored.

5) Blacklisting vs. Disabling a Device: Up to this point, the device is in a trusted state (in the upper half of Figure 2) meaning that other devices in the network will respond to its requests as well as sending their own requests for fragments the device possesses. If a device has not been detected within the network for an extended period of time, or is thought to be compromised, a message can be sent from the administrative account (signed with the administrative key), informing other devices within the system to "blacklist" the affected device. This ensures that missing devices can be automatically blacklisted if they leave the "active" area. Other devices will then ignore requests from the blacklisted device and will not send it file fragment requests. States below the horizontal dashed line in Figure 2 represent a device that has been "blacklisted" and is not currently trusted by the other devices in the system (but may still be accessed by a user). An example use case is when a soldier with a device is separated from his or her unit (and assumed captured). Such a "blacklisted" device may be reinstated when the administrator is able to verify that the device has not been compromised such as when the soldier rejoins his or her unit with the intact device.

In a more severe case, when a device is *known* (rather than suspected) to be compromised or lost, the administrative account can also attempt to remotely disable the device, similar to the BlackBerry standard security wipe feature [1]. This is represented by the center state in Figure 2. An example of this is a user explicitly reporting the device as lost. The difference in the two states (disabled vs. blacklisted) is that a disabled device is unrecoverable, e.g. may physically destroy its own hardware or securely wipe all initialization information listed in Section II-B1. To be usable again, a disabled device must be reinitialized through physical interaction, whereas a "blacklisted" device could potentially be reauthorized via the network, once the administrator has verified that the device is trustworthy. Note that this disabling operation is an extra cautionary feature to deal with devices which are *not*

under adversarial control — adversaries are assumed to be able to bypass this self-destruct mechanism prior to it being triggered. It does, however, allow us to prevent adversaries from obtaining functional devices which have been lost.

C. User

It is possible that a user's information (both PIN and token) is compromised, allowing the adversary to access the file system from any mobile device which is trusted by the network. The system handles this threat in a manner similar to that discussed for a compromised device, by revoking the user and token in question. When a user is suspected or known to be compromised, each device within the system is notified, and marks the PIN and key pair corresponding to that user and token as "untrusted." Requests made using that user's authentication information are ignored. As with devices, this notification is expected to be initiated by an administrative account If the user later is proven to be "trustworthy" then devices are notified and will once again respond to requests from that user. This type of revocation makes device compromise less than trivial even for adversaries skilled in bypassing tamper resistance technologies — there is now a time limit after which the device and/or token are useless to the adversary.

III. Evaluation

We evaluate our approach in two ways: we first examine the security improvements which the system offers, then we evaluate a prototype implementation to measure response time, computation overhead, and battery consumption.

A. Security Improvement

Given the context in which our proposed system will be deployed, and the possible capabilities of the adversary (a nation-state), we cannot reasonably expect to completely prevent the adversary from gaining access to the file system through a captured device. We do, however, wish to reduce the probability of this occurring or limit the timeframe in which the adversary can carry out attacks. Figure 3 shows the attack tree for adversaries attempting to access data files using the system. Prior work has either assumed devices are never captured, or that the adversary will be unable to break tamper resistance or the PIN. We consider much stronger adversaries, who can employ sophisticated techniques to crack device encryption, bypass tamper resistance of the device or the user token, and even capture targeted users to obtain their devices, tokens, and login information.

The left-most branch shows the case wherein a user is captured. In this case, the adversary likely gains access to both the device and the token. This leaves the PIN as the only feature keeping the adversary from accessing the system. The PIN is a far weaker secret than the cryptographic information held by the token, so even if the adversary is not able to get the user to reveal the PIN, it would only take a small amount of brute-force computation to crack it. We therefore assume that the probability of cracking the PIN is 1, and happens almost instantly. The unknown variable in this branch is the time required to realize the user, device, and token have been captured, and to blacklist the user and device, denying the adversary access to the system even if the device and token are physically compromised. If the probability of detection prior to the adversary cracking the PIN is 0.2, then that is the improvement to security that our system provides along this branch when compared to systems which do not provide the ability to revoke access to file fragments. In this scenario, detection of captured users is vital for security of the system.

The middle branch covers scenarios wherein the device and token are independently acquired by the adversary. For example, a device may fall out of a vehicle and the adversary can obtain it independently of a token, which might later be stolen from an authorized user. In this situation, it is imperative that soldiers report lost or stolen devices and/or tokens as soon as possible to limit the window of opportunity an adversary has to use these items to access the system. It should be noted that compromised tokens and/or devices can only be utilized on the network for which they have been initialized as other devices within the network must be able to recognize the signature of any token or device used to make fragment requests.

In the case on the far right, where a user is collaborating with the enemy, the adversary is assured access to the data. Our system has no effect on this attack, and insider attack protection is beyond the scope of this paper.

We compare the performance of our system with the existing systems, which do not implement revocation or multi-factor authentication, based on the probability that an adversary is able to access the system and thus access the distributed files. First, we utilize two equations to compare the relative probability of an adversary gaining access to a system with and then without our protocol being implemented. Then, we implement a simulation to show how our system performs compared to the other systems over time.

1) Calculations: In Equations (1a)and (1b), $P[c]$ is the probability that a user is captured with a device and token, $P[m]$ is the probability that a user is malicious, $P[d]$ is the probability that a device is obtained and physically compromised, $P[p]$ is the probability that a user's PIN is compromised, $P[t]$ is the probability that a token is obtained and physically compromised, and $P[nbl]$ is the probability that the compromised components are not blacklisted. We can use these equations to compare the likelihood that the adversary will gain access to a system using our design, versus systems using other designs.

$$P[a] = P[c] + P[m] + P[d] * P[p] * P[t] * P[nbl] \quad (1a)$$

$$P[b] = P[c] + P[m] + P[d] * P[p] \quad (1b)$$

Equation (1a) represents the probability that the adversary will gain access to a system implementing our design, while equation (1b) represents the probability that the adversary will gain access using a system not implementing our security improvements. It is easy to see that the strength of our solution lies in the difficulty of obtaining a token ($P[t]$) and in the quick response by the administrator to blacklist missing devices and tokens, reflected in the probability that the device and user are not blacklisted ($P[nbl]$). Note that for both systems, we take a conservative approach and assume that when a soldier is captured, the adversary *is* able to compromise the system. We do not allow for the possibility of either the device or

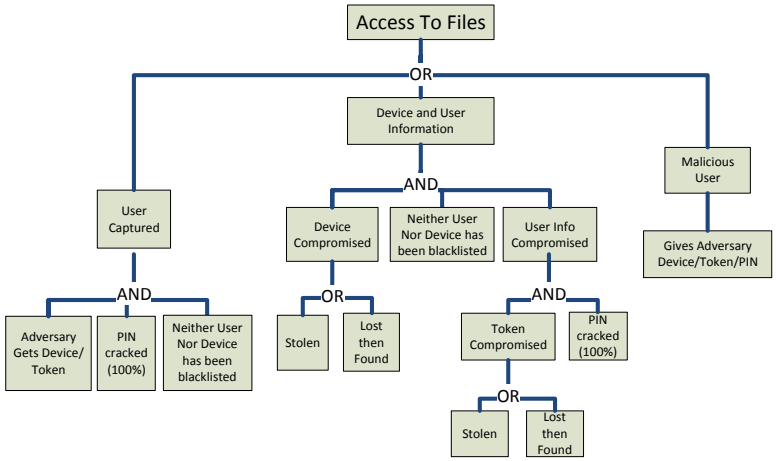


Fig. 3: Attack tree showing possible paths for an adversary to gain access to files within the network.

token being blacklisted before the adversary is able to do this. This represents the normal case for a system not implementing our security improvements and the worst case for a system with our improvements implemented. This leaves the middle branch, where the token and device are acquired independently, as the focus of our calculations. To evaluate the two systems, we make the assumptions shown in Table I for the range of each variable in the equations above.

TABLE I

Variable	Min	Max
$P[c]$	0.001	0.035
$P[m]$	0.0001	0.00015
$P[d]$	0.009	0.03
$P[p]$	1.0	1.0
$P[t]$	0.036	0.018
$P[nbl]$	0.01	0.02

These values represent our conservative estimate of what is expected in practice. It is important to note that the resulting probability of compromise is substantially reduced in systems implementing our design, as long as 1) the secret contained within the token is cryptographically secure, 2) the token is difficult for the adversary to obtain, and 3) the administrator is diligent in revoking access for devices and users that are compromised.

We performed calculations using both equations, with each variable being assigned a random value within it's estimated range and then calculating both P[a] and P[b]. This operation was repeated 1,000,000 times and the results were averaged to give the following values: $P[a] = 0.018$ and $P[b] = 0.0376$. This shows a 52% improvement in security by implementing our solution, even when assuming that a captured soldier will always result in a compromised system. In fact, the results with our security implemented are essentially reduced to the probability of a soldier being captured with a device and that device always being compromised since the median value for $P[c] = 0.018$. If we change our assumption to include the possibility that the device and/or token from a captured soldier can be "blacklisted" prior to the adversary compromising the system, the security improvement will become even greater.

2) Simulation: In order to better understand and visualize the benefits of our protocol, we developed a simulator to determine the probability of system compromise over time by an adversary who simultaneously gains access to a mobile device and token. The simulation iterates through slices of time, tracking the number of tokens and devices that are in each of 3 states:

- Possessed by a soldier
- Lost or stolen
- Blacklisted

Transition from one state to another is based on the probability of a device or token being lost or stolen during a given time slice and subsequently its absence being noted and reported to the other devices (which black list the device) during a later time slice. If, at any point in a simulation run, both a token and a device have been lost or stolen and neither has been blacklisted, OR a soldier is captured, the system is considered to be compromised and we record the time step during which this occurs (note that we assume conservatively that a missing token and device are in the possession of the same adversary).

Figure 4 shows the results of this simulation. The difference in the two curves shows the increase in security provided by our use of the token to provide multi-factor authentication both when a user connects to a device as well as when the user attempts to make requests for data file fragments from other devices. We performed 6000 iterations of this simulation, and record how often the system is compromised during each time slice. These values are divided by the total number of iterations (6000) to give the probability of a compromise occurring during a given time slice. Finally, the cumulative value of all probabilities prior to a given time slice are added to that slice's probability to give the probability that the system is compromised at or before a given point in time. The probabilities we use for the simulator are given in Table II.

These values differ from those used in the equations discussed in Section III-A1. In those equations, we are generalizing the probabilities for multiple devices and tokens over the span of an entire mission, while in the simulator, we account for each device and token individually and iterate over multiple segments of time. Thus, the probabilities used here are much

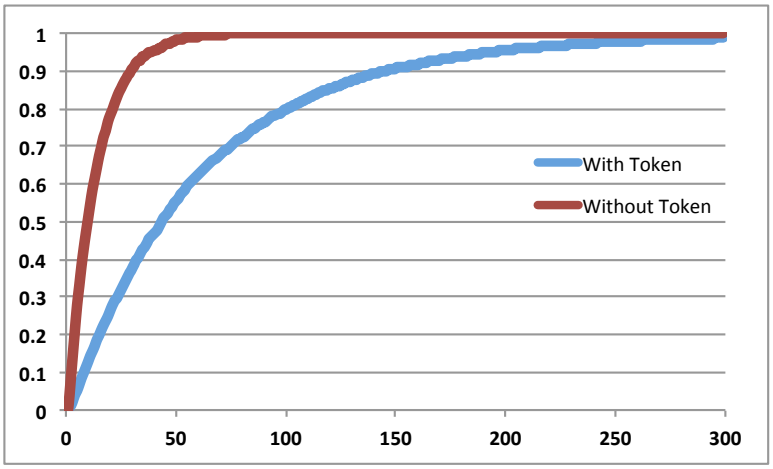


Fig. 4: Cumulative Distribution Function for the probability of an adversary compromising the system over time

TABLE II

Variable	Value
$Numberofdevices/tokens$	40
$P(soldiercaptured)]$	0.001
$P(devicelost/stolen)$	0.0008
$P(tokenlost/stolen)$	0.0012
$P(deviceblacklisted)$	0.50
$P(tokenblacklisted)$	0.85

smaller than those given in Section III-A1. These probabilities are our best conservative estimates of what these values would be in a real-world scenario.

We do not specify the units of time, as our main concern is to determine how the two configurations perform compared to one another. It is clear that during a reasonably large timespan (number of iterations), the system utilizing multi-factor authentication provides substantially stronger security than systems which do not.

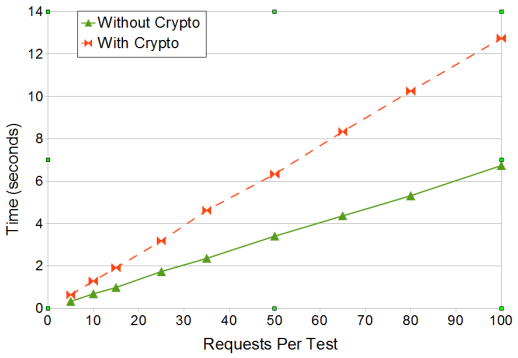
B. Implementation

We implement a prototype of our design in three components: one mobile device acts as the requesting device, a second acts as the responding device which holds the file fragment and responds to file requests, and a laptop acts as the wireless token. Each component of the system is described in detail below. In the deployed implementation, each mobile device will run at least 2 processes: one responds to incoming requests for local file fragments, and the other performs user authentication, makes requests to other devices, and handles reconstruction of data once the requisite number of fragments have been received.

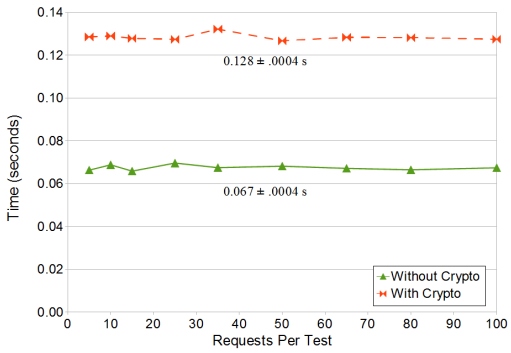
For testing, we used a Samsung Galaxy Tab 10.1 Tablets for both the requesting and responding mobile devices. These devices have a 1 GHz dual core processor, 16 GB of RAM and were running the Android version 4.0.4 operating system. We chose Bluetooth for communication with the token due to the wide availability of commodity hardware implementations and cross-platform API libraries. The token was simulated using a Java application running on a laptop. When launched, the application waits for a connection request over the Bluetooth socket and, once connected, continuously loops through a (receive challenge)/(sign challenge)/(respond) sequence. If the connection is broken or fails, the device returns to the initial state, waiting for a new socket connection. All mobile development used Java SE and the Android SDK [9] with the standard Java Bluetooth API.

Recall that when the user wants to access files, he or she must first authenticate (weakly) using a PIN. Once a valid PIN is entered, the mobile device retrieves the identifying information for that user's token and launches a separate thread to connect to it via Bluetooth. It then sends a challenge containing a current timestamp and the mobile device's ID to the token, which responds by signing the challenge. The device verifies the signature using the public key associated with the entered user PIN. An additional change required to comply with FIPS 201-1 is certificate verification [20] — we leave this for future work.

1) File Fragment Requests: In order to reconstruct a file, the requesting device retrieves the corresponding file fragments from other devices in the network. We use IP connections over Wi-Fi for inter-device communication in order to match our expected use case — two different modes of radio communication when using a dedicated wearable token. (Methods of managing wireless mobile networks have been studied extensively in the literature [3], [16], [8] and are outside of the scope of our work.) Once a secure socket is created between the two mobile devices, the requesting device sends information to prove that the user and requesting device are authorized to access the requested file, and that the token replied to a challenge from the requesting device. The responding device independently verifies the challenge-response data, including the signature from the token, and also verifies that the device ID in the challenge is the same as the ID of the device making the request. Additionally, since a timestamp was included in the challenge and device clocks were synchronized during initialization, the responding device can verify the freshness of the request. Finally, the signed copy of the request can be verified using the public key for the requesting device, stored by the responding device at initialization time. If all checks pass, the device returns the file fragment over the socket, otherwise the device drops the request.



(a) Time to complete the indicated number of requests



(b) Time per request

Fig. 5: Average time overhead: overall (a), and (b) per request, when using the token for additional authentication, with and without cryptographic overhead.

C. System Response

We measured the additional time overhead due to the signing and verification operations on the devices to ensure that users will not see significant increases in latency. While the device would typically send one request to each remote device containing a needed file fragment, each of our tests consists of 5 to 100 request/response cycles. This ensures that the times being measured are not dominated and masked by other operations, such as polling the token, which occurs in a separate thread. Response time is evaluated at the requesting device by measuring the time between generation of the request message and receipt of the response. We tested our system with and without cryptographic operations. For testing purposes, we used RSA with 1024-bit keys, but other algorithms might provide better security or performance metrics. While our implementations are software-only, we expect that dedicated cryptographic hardware will be used for the token and as part of the device, making our protocol even more power-efficient and further reducing latency [7], [15].

Figure 5a shows the results of our tests. Time overhead grows linearly with the number of request/response cycles in a given test run, but at different rates, reflecting the additional time needed for cryptographic operations. Both graphs show a Y-intercept near 0, indicating that our results are not significantly influenced by outside background operations of the device, or various radio noise and potentially competing Wi-Fi networks (otherwise the lines would cross the time axis above zero). Figure 5b shows that the average response time per request for each set of tests is consistent even while varying the number of requests per run. The average response time for the base case tests which do not implement cryptographic operations is 67 ± 0.4 milliseconds — unnoticeable to a user.

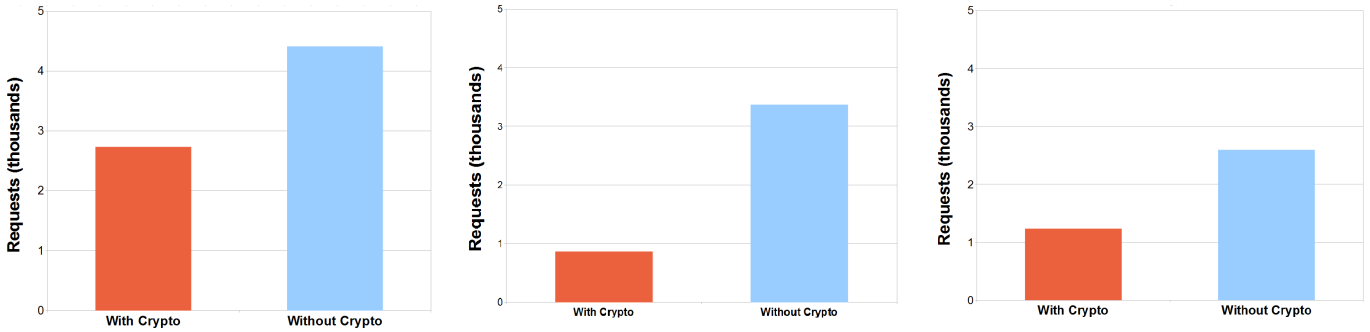
In our second set of tests, we enabled cryptographic operations on both the requesting and responding devices as described in Section III-B. As Figure 5b shows, the average response time increases to an overall average of 128 ± 0.4 milliseconds. Based on this configuration and the tests we performed, the cryptographic operations add approximately 61 milliseconds to the request/response cycle. Note that although this may be noticeable for a user, delays do not increase linearly with the number of requests under normal operation. A portion of the delay occurs at the responding device, and since many devices will be queried simultaneously for file fragments, these delays occur in parallel after the requests are sent. Considering a mobile wireless network with multiple devices distributed throughout a military unit's operating area, the expected round-trip transmission times to recover the file fragments will dominate, so the cryptographic delay does not contribute significantly to latency.

D. Battery Consumption

Given the context in which our system is expected to be deployed, it is crucial that consumption of battery power be considered in any decisions that are made concerning mobile device operations. For that reason, we performed a set of tests to determine the expected change in battery consumption for both the requesting and responding devices when implementing our system. We tested battery consumption for file requests in a manner similar to the response time tests above. Instead of performing a set number of request/response operations, we repeated these tests until the battery level dropped by a given percentage and then we compare the results found with and without the cryptographic operations being implemented.

1) Requesting Device: For the requesting device, we repeated these tests until the battery level dropped by 3%. This test was performed multiple times, both with and without cryptography. Figure 6a shows the average number of requests required to deplete the battery by 3%. Without cryptography, the average number of requests was 13245 ± 461 (4415 requests per 1%) , and cryptography reduced this to 8209 ± 257 (2736 requests per 1%).

Tests not incorporating cryptographic operations showed greater variability (suggesting interference from background processes). While the battery drain is naturally increased when using cryptography, the number of requests in our tests was much greater than we would expect in practice. Per-request power consumption is quite small when the cryptographic operations are enabled. The battery lifetime of a requesting



(a) Galaxy Tab Requesting device (b) Galaxy Tab Responding device (c) Droid Pro Requesting Device

Fig. 6: Number of requests needed on each device to drain battery power by 1%

device would be entirely dependent on the number of file accesses a user makes as well as the size and number of fragments required to reconstruct whole files, where there is a direct trade-off between battery consumption and security — more required fragments means the adversary is less likely to learn any information by compromising devices only, but requires a higher rate of battery depletion to reconstruct files.

2) Responding Device: Next, we tested battery consumption on the responding device, with results shown in Figure 6b. Again, we performed these tests until a battery drain of 3% was measured. With cryptography enabled, 2601 ± 42 requests consume 3% of the tablet's battery power (867 per 1%). Without cryptography, 10094 ± 382 requests are necessary to cause a 3% drop (3365 per 1%).

There is a greater increase in battery consumption on the responding device than on the requesting device. This would reflect the extra verification that is required on this device. Likewise, the overall cost per request is higher on the responding device. Battery consumption for responding devices will be dependent upon what file fragments a device is storing. If a device possesses fragments from 'popular' files which are accessed often, then that device is likely to respond to more fragment requests than a device which stores less 'popular' fragments. Additionally, the use of a larger number of fragments will require a greater number of devices to respond when a given user wants to access a file.

3) Additional Requesting Device: Given that the tablets utilized in our testing have more powerful batteries than typical handheld mobile devices, additional tests were performed using a Droid Pro mobile phone as the requesting device. This device should more closely resemble the type of device a soldier might carry as well as demonstrate the variability in performance that might occur on different devices. The battery consumption tests were performed exactly as described above with one exception. Since the API for this device only provides battery measurements in 10% increments, the tests were run until a drop of 10% was measured. As expected, it required fewer requests to realize a 1% drop in battery power using this device. The tests incorporating cryptographic operations averaged 3708 $\pm$ 53 requests for a 10% drop (371 per 1% drop), while the tests without cryptography averaged 7789 $\pm$ 182 requests for a 10% drop (779 per 1% drop).

4) Token: Assuming the token is self-powered, and generalizing from worst-case power consumption results (requesting device), and given a 30 second polling interval for the token, we would drain 1% of the token battery in over 3 hours of continuous operation, allowing for more than 12 days of use before recharging. Since the token will only be performing a single operation (signing the received challenge), performance is expected to be much better per cycle than what was found for the requesting device. Additionally, given that the token is expected to be composed of dedicated, specialized hardware, battery life can be greatly prolonged with relatively little monetary investment [14], [12], [17].

IV. Related Work

The National Institute of Standards and Technology has published a standard that addresses proximity-based authentication for mobile devices. The authors describe the use of Bluetooth tokens for user authentication to a single mobile device, with a long-term pairing between the mobile device and token (paired at initialization) [13]. However, there is no discussion of extending this security to the application level. In [19] and [4], the authors also suggest a wearable token to authenticate a user, but their system is limited to letting the computer or mobile device detect when the token has moved too far away, and logging the user out or locking the device. Our system allows users to connect to different devices within the system as needed, without long-term explicit pairing, and utilizes the authentication information provided by the token as a means to authenticate individual file fragment requests made to other devices.

Tahoe-LAFS is a free, open source system for distributed federated data storage [26]. Users are issued capabilities (a copy of the key to a given ciphertext) to specific files which they can then delegate to other users by sharing the key with them. Tahoe-LAFS uses erasure coding of files across multiple servers to provide data resiliency and pre-erasure coding encryption for confidentiality. This system puts the burden of access control and user management on the owners of the data — individual system users — through capability delegation. One compromised user is able to share irrevocable access with any number of additional users. The number of servers, however, is limited, so the system is not peer-to-peer and is a poor choice for ad-hoc wireless networks.

MDFS is a distributed file system for mobile networks built for security of files with the assumption that an adversary can gain possession of a device and access local storage [10]. It is intended for military use and provides data replication and secrecy using encrypted then erasure-coded files distributed across multiple mobile devices in an ad-hoc peer-to-peer network. The encryption key for each file is split using Shamir secret sharing [25], with key fragments distributed across multiple devices as well. This system provides resilient data storage, and limited protection of the data in that an adversary is unable to recover any information from the data file fragments stored on a single device. However, the authors do not attempt to address restricting access to files on the system from a compromised device. Anyone possessing a device that is recognized as part of the network is able to obtain access to any of the shared files once they have cracked the PIN required for logging into the device.

V. Conclusion

We reduce the risk posed by an adversary who is trying to gain access to confidential files stored within a mobile wireless network in a combat setting. By utilizing multi-factor authentication we alleviate the problem of full network access from logged-in devices and provide revocation capabilities for users and devices independently. Our system provides a significant improvement in security over MDFS, without increasing user interaction complexity or the length of PINs. Our system is flexible enough to allow users to operate any device on the system, but still limits an adversary's window of opportunity to use a device that is stolen or found after being lost. None of these benefits requires improved tamper resistance.

Our tests show a slowdown of .061 seconds on average for each file fragment request – barely noticeable to a user. While the delay could become more pronounced if there are large numbers of file fragments that must be retrieved, these requests are pipelined and parallelized across multiple communication paths so the user-perceived delay is minimized. Evaluation on real-world devices also shows that per-fragment request battery consumption nearly doubles due to cryptographic overhead, but even with this increase, our test requesting device make over 2730 fragment requests at the cost of 1% battery consumption, and our responding device can service more than 860 requests to cause 1% battery power drop. Therefore, practical costs of implementing our proposed security strategy are reasonable, especially given the security benefits.

There are numerous opportunities to expand this work, including testing various device-token communication protocols and evaluating designs for dedicated crypto-tokens, which have become quite inexpensive [7], [15]. As noted, we did not implement certificate verification, which must be added to meet FIPS 201-1 specifications. Additionally, investigating potentially faster and more power-efficient cryptographic algorithms such as elliptic curves may help improve system response times and reduce battery consumption [24], [6], [21], [5].

References

[1] Blackberry. Erasing file systems on BlackBerry devices, 2012. [Online] http://docs.blackberry.com/en/admin/deliverables/4322/Erasing%20file%20systems%20on%20BlackBerry%20devices%20-%204.1.6%20-%20Technical%20Overview.pdf.

[2] Bluetooth SIG. Bluetooth specification version 4.0, 2010. [Online] https://www.bluetooth.org.

[3] Candolin, C., and Kari, H. A security architecture for wireless ad hoc networks. In *MILCOM* (2002).

[4] Chen, Y., and Sinclair, M. Tangible security for mobile devices. In *Mobiquitous* (2008).

[5] Eberle, H., Wander, A., Gura, N., Chang-Shantz, S., and Gupta, V. Architectural extensions for elliptic curve cryptography over GF(2^m) on 8-bit microprocessors. In *ASAP* (2005).

[6] English, T., Keller, M., Man, K. L., Popovici, E., Schellekens, M., and Marnane, W. A low-power pairing-based cryptographic accelerator for embedded security applications. In *SOCC* (2009).

[7] Feldhofer, M., Dominikus, S., and Wolkerstorfer, J. Strong authentication for RFID systems using the AES algorithm. In *CHES* (2004).

[8] Garg, N., and Mahapatra, R. Manet security issues. *International Journal of Computer Science and Network Security 9*, 8 (2009).

[9] Google. Android SDK, 2012. [Online] http://developer.android.com.

[10] Huchton, S. Secure mobile distributed file system. Master's thesis, Naval Postgraduate School, 2011.

[11] Hughes, G. personal conversation.

[12] Hwang, D., Lai, B.-C., Schaumont, P., Sakiyama, K., Fan, Y., Yang, S., Hodjat, A., and Verbauwhede, I. Design flow for HW/SW acceleration transparency in the thumbpod secure embedded system. In *Design automation conference* (2003).

[13] Jensen, W., Gavrila, S., and Korolev, V. Proximity-based authentication for mobile devices. In *International Conference on Security and Management* (2005).

[14] Matsuoka, Y., Schaumont, P., Tiri, K., and Verbauwhede, I. Java cryptography on kvm and its performance and security optimization using hw/sw co-design techniques. In *CASES* (2004).

[15] McLoone, M., and Robshaw, M. Public key cryptography and RFID tags. In *CT-RSA* (2006).

[16] Michalakis, N., and Kalofonos, D. Designing an NFS-based mobile distributed file system for ephemeral sharing in proximity networks. In *ASWN* (2004).

[17] Nambiar, V., Khalil-Hani, M., and Zabidi, M. Accelerating the AES encryption function in OpenSSL for embedded systems. In *ICED* (2008).

[18] NFC Forum. Near field communication specification, 2012. [Online] http://www.nfc-forum.org/specs/.

[19] Noble, B. D., and Corner, M. D. The case for transient authentication. In *ACM SIGOPS European workshop* (2002), EW 10.

[20] of Commerce, U. D. Fips 201-1, 2006. [Online] http://csrc.nist.gov/publications/.

[21] Oliveira, L., Aranha, D., Morais, E., Daguano, F., Lopez, J., and Dahab, R. TinyTate: Computing the Tate pairing in resource-constrained sensor nodes. In *NCA* (2007).

[22] Osborn, K. Smart phones increase 'SPOT' reporting in Army evaluations, June 2011. [Online] http://www.army.mil/article/59435.

[23] Pendl, C., Pelnar, M., and Hutter, M. Elliptic curve cryptography on the WISP UHF RFID tag.

[24] Scott, M., Costigan, N., and Abdulwahab, W. Implementing cryptographic pairings on smartcards. In *CHES* (2006).

[25] Shamir, A. How to share a secret. *Commun. ACM 22*, 11 (1979).

[26] Wilcox-O'Hearn, Z., and Warner, B. TAHOE: The least-authority filesystem. In *StorageSS* (2008).

[27] ZigBee Alliance. ZigBee 2007 specification, 2007. [Online] http://www.zigbee.org.