GROK SECURE MULTI-USER CHAT AT RED FLAG 2007-03*

Roger Khazan, Joseph Cooley, Galen Pickard, Benjamin Fuller

MIT Lincoln Laboratory
244 Wood Street, Lexington, MA 02420-9108

ABSTRACT

This paper describes the GROK Secure Chat experimental activity performed by MIT Lincoln Laboratory at USAF Red Flag 2007-03 exercises and its results.

INTRODUCTION

MIT Lincoln Laboratory, Information Systems Technology group, under sponsorship from the Air Force Cryptographic Modernization Program Office (AFCMPO), is developing a usable solution for securing communication and collaboration of ad-hoc dynamic groups of participants in mobile tactical environments. Two main software components being developed in this project are 1) a generic software library, called GROK, for efficient and decentralized creation, dissemination, and use of group keys; and 2) a secure text-chat application that uses GROK to provide end-to-end confidentiality to multi-user chat rooms; in this document, we refer to this application as GROK Secure Chat or GSC. The project started in FY06 and is currently in its research, development, and demonstration phase.[1]

In August of FY07, MIT LL participated in USAF Red Flag 2007-03 (RF07) exercise, held at Nellis Air Force Base, Nevada[2]. MIT LL's primary mission was to collect C2ISR images from the MIT LL's Paul Revere 707 aircraft using Lincoln Multi-mission ISR Testbed (LiMIT) and forward these images from Paul Revere to the Nellis Combined Air Operations Center (CAOC-N) in near-real-time.

In addition to this primary mission, MIT LL ran three data collection and experimentation activities: 1) Network Control Plane; 2) Service Oriented Architecture; and 3) GROK Secure Chat. The first two are covered in separate MIT LL reports [2]. This paper describes the GROK Secure Chat activity and its results.

Goals: The main goal of this activity was to perform experimentation and testing of GSC in a real use-case setting. We aimed to accomplish the following:

- Capture experimental data associated with GSC and the underlying network.
- Identify any implementation and configuration problems that may become visible under normal use of GSC in a disadvantaged network;
- Observe use-patterns, identify any usability problems, and develop ideas for improving usability;
- Feed all of the above into future R&D effort and help improve future GSC deployments and exercise participations.

Summary: The GROK Secure Chat activity was successful in achieving its goals. GSC was used by the MIT LL 4xComm team as the sole collaboration tool for exercise management. In addition to allowing us to evaluate GSC performance under real-use conditions and to identify opportunities for future R&D improvements, our participation at RF07 provided us with a real-use data corpus to drive future experimentations and evaluations of GSC in a network test-bed environment.

RF07 AND NETWORK OVERVIEW

RF07 was a two week event: the first week MIT LL performed system integration and testing; the second week MIT LL flew four mission flights, Monday through Thursday. During the second week, each mission-day began at 1300, the flight began around 2100 and lasted four to six hours. Flight details were optimized for LiMIT image collection, MIT LL's primary mission. The other experiments, including GROK Secure Chat, were run on a non-interfering basis.

MIT LL operated four sites: MIT LL's Paul Revere (PR) EC-707 aircraft, a trailer (TR) located at Nellis AFB, a portable node (PO) located at a Black Mountain (BM) remote facility, and a workstation located at Hanscom AFB (HA). Each of the sites hosted a sophisticated local area network comprised of a number of service-specific ma-

[1] Additional information can be requested from the MIT LL POC, Dr. Roger Khazan, rkh@ll.mit.edu, and from the AFCMPO PM, Garth N. Nelson, CPSG/ZX, garth.nelson@lackland.af.mil.

[2] Red Flag is an advanced aerial combat training exercise. [Its] purpose is to train pilots from the U.S., NATO, and other allied countries for real combat situations [1].

* This research was sponsored by the United States Air Force under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are not necessarily endorsed by the US Government.

chines and operator laptops. The sites were interconnected with a combination of wired, Line-Of-Sight, and Beyond-Line-Of-Sight links (Figure 2 and Figure 3). This combination of links was used to communicate LiMIT images in near-real-time from PR to TR; after being received at TR, the images were relayed to the Nellis Combined Air Operations Center (CAOC-N) via a direct land line. Each of the sites was manned by a number of operators: PR and TR had six-to-ten each, BM – two and HA – one.

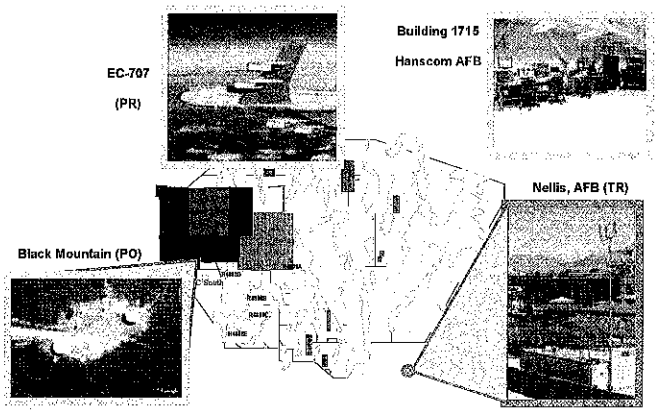

Figure 1. Four sites operated by MIT LL at RF07.

A more detailed description of the RF07 exercise and the network operated by MIT LL at RF07 can be found in the *Red Flag 07-3 Report* from the 4xComm Tech Team [2].

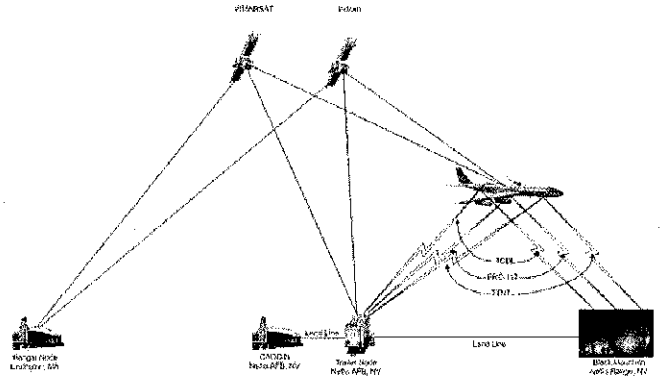

Figure 2. Different communication links used to connect the RF07 sites.

GROK SECURE CHAT USE OVERVIEW

GSC was deployed at three sites: PR, TR, and PO and was used by the 4xComm team as the sole communication tool. Figure 4 shows a screenshot of an operator's computer monitor taken at a mission flight. A GSC window on the left contains a snippet of conversation among ten operators; the lock icons surrounding text are GSC's message security indicators. The two windows on the right are a 4xComm application indicating which wired and wireless links connected the different sites at that moment.

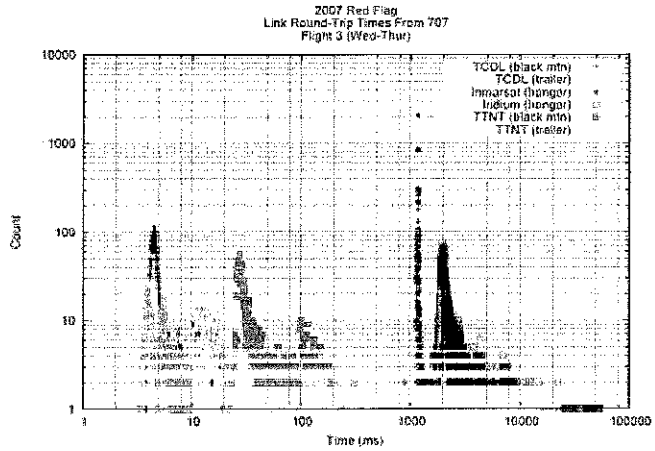

Figure 3. Round-trip link delays, computed using repeated pings (ICMP echo requests/responses).

Figure 5 summarizes the use of GSC during the Tuesday, Wednesday, and Thursday mission flights; it shows the numbers of hours, users, and messages exchanged using GSC. Cumulatively, during these three days, GSC was used for ~20 hours, hosted ~20 different users, and helped exchange ~2500 chat-room messages.

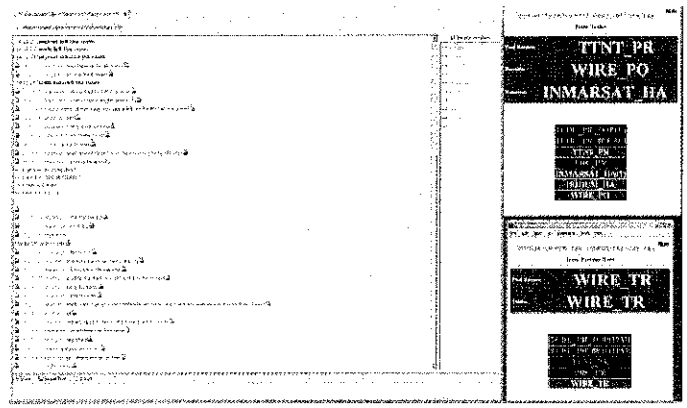

Figure 4. A screenshot showing GROK Secure Chat and a link-type-monitor utility.

In addition to the main chat-room ("Network"), GSC was deemed "safe" to be deployed in the Test Directors (TD) chat-room on Thursday. According to the 707 TD, Capt. Dwyer, the introduction of security and the use of GSC were transparent to the TDs.

One of the salient GSC features is its ability to seamlessly accommodate unplanned membership changes. This feature was useful during the Thursday mission, when two L3

engineers were able to join "on-the-fly" the Networking chat-room in order to support the CDL link. (On the screenshot above, "bmcdl" and "trlcdl" are their user-names; "bm" stands for Black Mountain, and "trl" stands for "trailer"). Transparently to the room participants, GSC rekeyed everyone when the L3 engineers joined and when they left the room, thereby limiting their access to communication to the exact time they were allowed to participate in the room.

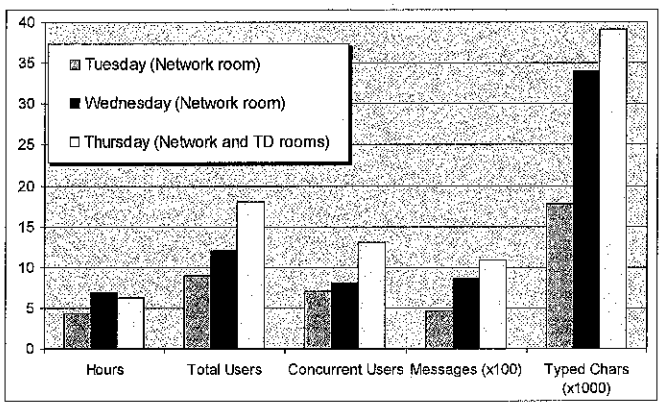

Figure 5. A summary of GSC use at RF07.

Two caveats: 1. the analysis does not include the Monday mission because the mission was cut short due to weather conditions and because for most of this short mission a chat server misconfiguration prevented chat between TR and PR. 2. GSC was not deployed at the HA workstation because of our inability to upgrade the workstation's system software to the required versions; instead, GSC was modified to provide an insecure chat bridge to the operator at HA.

GSC SYSTEM SET-UP AND DATA COLLECTION

Chat system: The chat system underlying GSC is comprised of two popular open-source products: the OpenFire chat server, which implements the Jabber XMPP protocol, and the Pidgin chat client, which supports Jabber XMPP protocol as well as other chat protocols [3]. Chat servers host user accounts and chat-rooms. Users logged in to one server can join chat rooms hosted on another server, assuming such access is enabled and the users have sufficient permissions. When a user sends a message in a chat room, the message goes from the user's chat client to the chat server hosting the chat room; the server then forwards a copy of the message to the chat clients of each of the room's members.

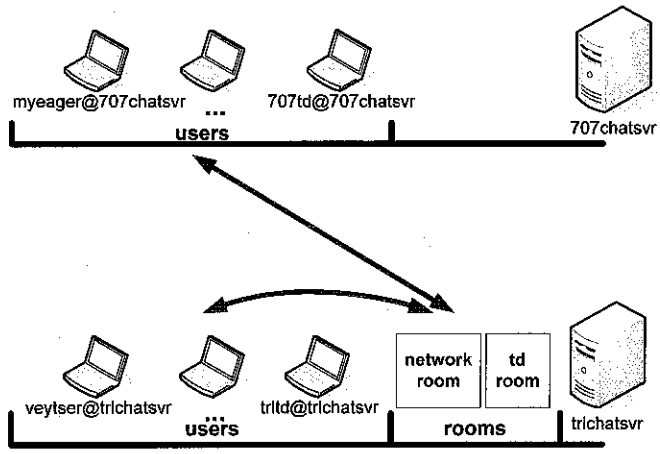

Figure 6. Chat system set-up at RF07.

At RF07, MIT LL operated two OpenFire chat servers, `trlchatsvr` and `707chatsvr`, at TR and PR. Operators at these sites connected their Pidgin chat clients to their local servers; PO operators connected to the TR chat server. After logging in to their local servers, operators joined and participated in the "Network" chat room, which was hosted at TR (Figure 6). TR and PR typically had half a dozen chat operators online during mission flights; PO typically had one. TR also hosted the "Test Directors" chat room, which was used by the TR and PR test directors.

Without GSC, this same Pidgin/OpenFire chat-system configuration would have been used at RF07. The only difference would have been the lack of end-to-end confidentiality and authenticity for communicated messages that GSC provided.

Secure messaging: GSC is implemented as a Pidgin plugin. The plugin encrypts outgoing messages and decrypts incoming messages using GROK, MIT LL's general-purpose software library for group keying, encryption, and authentication. Messages sent in a chat room are encrypted in such a way that only current members of the chat room can decrypt them; instant messages sent to a specific user are encrypted in such a way that only the sender and that user are able to decrypt them. Thus, security provided by GSC is end-to-end; this type of security is similar to that of secure email (S/MIME).

GROK encrypts chat messages using a standard, symmetric-key encryption algorithm called AES (Advanced Encryption Standard); this encryption is fast and has small message-size overhead. When group memberships change, new group keys are encrypted to the chat room members and piggybacked onto encrypted chat messages. GROK implements two group keying methods: Public-Key Group Encryption (PKGE) and Dynamic Set-Key Encryption

(DSKE). PKGE is based on modern, so called Bilinear-Map, cryptography and is an extension of a broadcast encryption scheme by Boneh et al. [4,5]; DSKE is an original method based on traditional, standardized cryptography. Messages carrying group keys are called "rekey" messages; they are typically larger than "normal" messages because they include encrypted group keys and digital signatures of the senders, in addition to encrypted chat messages (Figure 7).

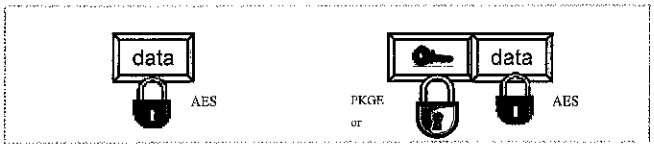

Figure 7. A depiction of normal (left) and rekey (right) messages. Rekey messages carry encrypted group keys in addition to AES encrypted data. The multicolored lock depicts group encryption.

At RF07, we experimented with both methods: PKGE – on Wednesday in the Network room and Thursday in the Test Director's room; DSKE – on Tuesday and Thursday in the Network room.

The presence of GSC and the choice of PKGE and DSKE are largely transparent to users, aside from the security icons surrounding each message: closed locks indicate GSC secured messages; open locks indicate messages sent in plaintext; non-decryptable messages are displayed as a special warning icon and a phrase "GROK Secure Message." Under normal circumstances, GSC should be able to decrypt all messages received by intended recipients. However, decryption may fail because of expired credentials, missing encryption keys, software bugs, or when a chat client receives an encrypted message that was not intended to the client's users; the latter may happen when message histories are replayed by chat servers.

The GSC system is a prototype, not a mature system. Because we did not want potential GSC problems to adversely affect other MIT LL's missions, we modified the GSC Pidgin plugin with the following operational fall-back: a plaintext copy was included with each ciphertext message; if decryption failed for any reason, the plugin displayed this plaintext copy and the warning icon. This way we could carry out GSC experimentation in the real-world, mission-critical setting, and at the same time avoid possible GSC problems getting in the mission's way. The presence of this operational fall-back did not interfere with the achievement of our RF07 goals, which we describe in the next three sections.

Logging: In order to centralize data logging and simplify deployment of the system-under-test, all Pidgin clients at each site were run as separate processes on the same "logger" machine, called `logbt`, `logba`, and `logbm` at TR, PR, and PO respectively. Operators interacted with their Pidgin clients from their laptops via X-Windows-forwarded SSH sessions; to operators, it appeared that Pidgin clients ran directly on their laptops.

The GSC plugin and the instance of GROK running at each Pidgin client recorded time and size information for each message at different processing stages: sent, encrypted, received, decrypted. A hash-based message identifier was included with each recording to enable identification of same messages at different processing stages and in different users' logs. Other information about internal GROK processing was also recorded. In order to simplify post-experimental analysis, client-server connections between Pidgin clients and OpenFire serves were not secured with Transport Layer Security (TLS).

EXPERIMENTAL DATA AND ANALYSIS

Following RF07, we have been analyzing the collected data; this is done using scripts (Perl, Python, and Bash), UNIX utilities, MATLAB, and Excel. Here, we focus on the data collected during the Wednesday flight (PKGE), as it corresponds to the cleanest sample of GSC's use by the 4xComm team at RF07. Data collected on Tuesday and Thursday were tainted with our testing and experimentation and, though invaluable for our internal R&D, are not a fair representative of GSC's expected performance. To summarize:

- Delay overhead (OH) due to GSC crypto computations was negligible compared to link latencies and message composition times (the latter were not measured).
- An average XMPP message was around 450 bytes as transmitted; this was not a problem for satellite links.
- However, the sizes of typed messages were small, only 30-40 characters on average. Majority of the message-size OH was due to XML/XMPP encoding of these messages -- about 60%. Later in the report, we estimate a possible reduction of this OH, and hence the overall message sizes, with XMPP compression. XMPP compression is supported by OpenFire but not the current version of Pidgin.
- Most messages were normal, i.e., non-rekey. GSC crypto OH for these was 60-80 bytes, similar to that of IPsec (49 bytes) and TLS (54 bytes), two popular secure channel protocols. Additionally, binary ciphertext had to be Base64-encoded in order to be sent via XMPP, resulting in ~0.33 size-expansion. XMPP messages were also tagged in a way to be transparent to non-GSC chat clients (39 bytes). The overheads from

both Base64 encoding and tags could have been close to eliminated with XMPP compression.

- The number of group changes and rekey messages (excluding our own testing) was very small in comparison to normal messages; this was expected. GSC OH of rekey messages varied from eighty to several hundred bytes, but was much smaller than the OH of the alternative S/MIME and WS-Security methods (in the order of kilobytes).

We now present a more detailed sampling of the analysis results.

Delay overhead: The bar-chart on Figure 8 shows delay measurements due to encryption, network travel, and decryption times for ten consecutive messages sent by a PR operator in the Network room during the Wednesday flight. The messages traveled to the TR chat server and then were sent to all the room members, including the sender. All the time measurements were done at the sender, using the same computer clock.

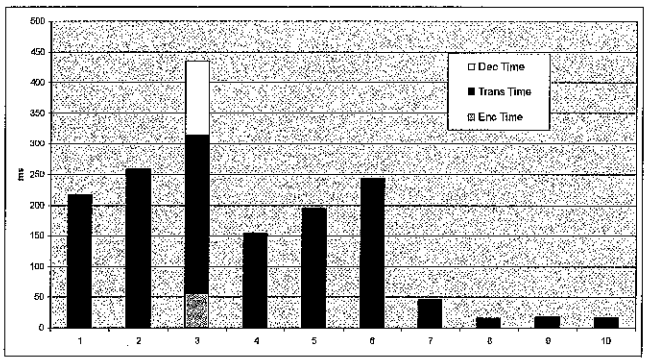

Figure 8. Delays due to encryption, round-trip transmission, and decryption (PKGE, Wed). Crypto times, except for those of the rekey event, are dwarfed by transmission delays.

In the middle of this sequence, connection between PR and TR was switched by 4xComm from a high-latency link to TCDL. The third message was a rekey triggered by a user joining the chat room. Encryption and decryption times were 0.2—0.4 ms for normal messages (not visible on the chart), and 56 ms and 121ms respectively for the rekey message; recall, that these were computed on a logger machine (`logba`), whose decryption and logging resources were concurrently shared among all PR users. Initial link round-trip time was around 200 ms (either over TTNT in both directions or over TTNT in one and TCDL in the other), and 5—45 ms over TCDL. The chart suggests that, the times taken by cryptographic operations, aside from the rekey events, are dwarfed by communication latency.

Message-size overhead: The bar-chart on Figure 9 shows different sources of message-size overhead for all the messages sent during the Wednesday mission.

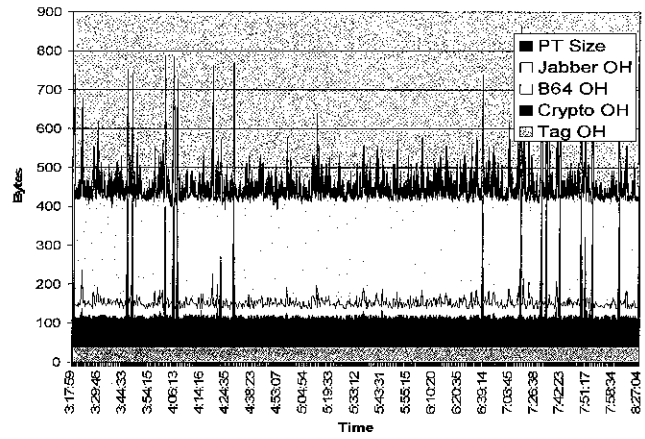

Figure 9. Message-size overheads for all GSC messages sent during the Wednesday flight.

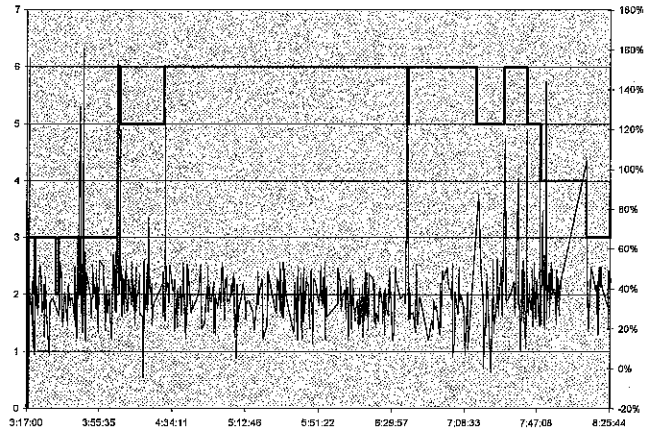

Figure 10. Number of people in the Network chatroom (black line, left axis) and the relative message-size OH (blue line, right axis) from using GSC encryption versus not, during the Wednesday flight.

PT Size is the number of bytes in the user-typed chat messages. Jabber OH is due to XMPP formatting of the chat messages, which includes a number of XML fields, including the sender's username and the destination chat-room name; this was ~275 bytes. Crypto OH is due to GROK/PKGE; it is the difference between ciphertext and plaintext sizes. For normal messages it was ~75 bytes, and for the rekey messages it was ~200 bytes. B64 OH is the overhead of encoding ciphertext into Base64 ASCII codeset, so it can be sent via XMPP. Tag OH is due to special XMPP tags (39 bytes) that we use to hide ciphertext from non-GSC chat clients. Note that the operational fall-back described in the previous section is excluded from this and the other analyses shown here.

GSC size OH: Figure 10 combines two charts: a) the membership changes in the Network chatroom (black line,

left axis), and b) the size OH added by GSC to every message relative to the size that would have occurred had GSC not being used (blue line, right axis). For the most part, the chatroom hosted half a dozen 4xComm operators during the Wednesday flight (not counting "test" personas: alice, bob, carol, etc.). GSC message-size OH was approximately 15-30% when room membership stayed constant, which was most of the time. During the few membership changes, the OH increased to 100-150% due to GSC re-keying.

Without the GSC plugin, Pidgin's XMPP messages encode users' plaintext twice, once in rich format using XML (e.g., specifying font and color) and also unformatted using plain ASCII characters. With the plugin, XMPP messages carry only single copies of ciphertext. Thus, Pidgin produces smaller XMPP messages with GSC than without when the size of unencrypted text is larger than the crypto OH added by GSC. This can be observed in several messages in Figure 10, where the OH falls below 0%.

Cumulatively, during the Wednesday mission, GSC was used for 6.9 hours; there were a total of 12 users, 9 of which were concurrent; 861 messages (34kB of text) were sent. Most use of GSC happened during the 4 hour flight: 708 messages sent, 12 of which were rekey –this means that only 12 unique group configurations were seen during the flight. The pie chart on Figure 11 shows the breakdown of size averages into five categories: plaintext, cryptographic overhead, Base-64 encoding overhead, Jabber overhead, and tag overhead.

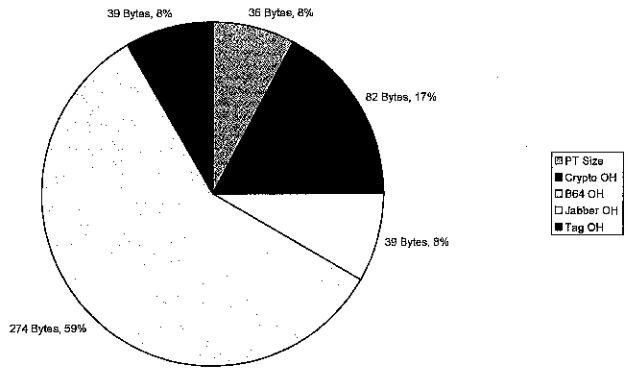

Figure 11. Breakdown of message-size OH averages by overhead-source for the Wednesday mission.

XMPP Compression Analysis: An obvious remedy to the large XMPP (Jabber) OH is data compression. The XMPP standard actually allows for XMPP compression, though Pidgin currently does not implement it. To study message-size reduction possible with XMPP compression, we applied zlib compression to the RF07 captured data in a way consistent with XMPP compression: each XMPP stream was compressed one packet at a time with respect to the prior packets in the stream. The compression ratio, i.e. the ratio of uncompressed data to compressed data, was around 3; in other words, with XMPP compression we expect each packet to be on average a third of the size observed at RF07. This compression occurs because different packets have very similar XMPP formatting and because the ASCII-based Base64 encoding is re-encoded into full binary data. For comparison, we computed the compression ratio for all the packets after transforming them to what they would have been without GSC being enabled. The compression ratio was expectedly higher, around 7, because encrypted payloads were replaced with plaintext data, which compresses well, unlike random ciphertext.

PROBLEMS AND TAKEAWAYS

Here we give a sampling of the types of problems and takeaways we identified thanks to our participation at RF07.

- Thanks to using GROK at RF07 in a real, complex setting, we were able to identify a number of intricate software problems; these have been since analyzed and fixed.
- During the first day, some users joined chat without GSC being enabled in their Pidgin clients. Group keys transmitted by others were missed by these clients, so later, when GSC was turned on, GSC lacked these keys and was unable to decrypt messages. The good news is that prior to RF07 we invented a special protocol, called Authenticated Statement Exchange (ASE), that, when implemented, will solve this problem. With ASE, missed keys would be retransmitted.
- Unrelated to GSC, we observed a number of shortcomings associated with the use of XMPP in tactical networks. The first one is the large XMPP-size overhead – This can and should be solved with XMPP compression; we recently extended Pidgin to support XMPP compression. The second one is the way chat-rooms are implemented in XMPP: the server hosting a chat room sends a separate copy of each message to each of the current members; thus, multiple copies of the same messages were sent from the TR chat server to the users on PR. A solution to this problem requires an extension of XMPP to support distributed chat-rooms, similar to the Robust Chat system developed by MIT LL several years ago (Khazan, et al, [6]).

USABILITY OBSERVATIONS

In addition to collecting data, logging problems, and identifying ideas for future R&D, we observed how users interacted with GSC and solicited their feedback about GSC.

- We were happy to see that, for the most part, the use of GSC with the Pidgin client was transparent to the users.
- Interoperability between GSC and non-GSC chat clients was important. Messages sent by non-GSC clients were displayed on GSC clients as "unsecured" messages, in red and surrounded by open-lock icons. Encrypted GSC messages received by non-GSC clients showed up as the phrase: "GROK secure message", instead of Base64-encoded ciphertext. To implement this feature, we used special XML tags to hide GSC ciphertext from non-GSC clients. In the future, the message-size OH from these tags (39 bytes) can be close to eliminated with XMPP compression.
- When compared to a different chat client used by operators in CAOC-N, we found Pidgin significantly more usable. Usability and other features are generally better in systems that have a tight feedback loop from users to developers, such as open-source systems; being widely-used, actively-developed, and continuously-improved also helps.
- We thought that it would be useful to extend the GSC user interface (UI) with a button for users to submit their comments and feedback to us. It would also be helpful if, during test deployments, any problems were automatically forwarded to us, similarly to how this is done by commercial companies such as Microsoft, Google, and Apple.
- Another useful GSC UI extension is a check-box button next to the text-input window for enabling encryption and digital signatures on per message basis. This feature has been recently implemented.
- Chat servers can be enabled to replay all or partial message histories to joining users. However, this feature violates end-to-end security and is incompatible with GSC: GSC clients are able to decrypt only those messages (historical or otherwise) that were explicitly encrypted to them. Our experience at RF07 and past exercises suggests that history replay is largely unnecessary and is ill-advised in low-bandwidth, high-latency networks. This said, it is foreseeable that a history feature can be useful in certain settings; if so, its implementation has to explicitly uphold the desired security policies governing access control.
- GSC requires users to have personal digital certificates, which are used for user's identities and digital signatures. Deployability and usability can be improved by implementing a certificate bootstrapping solution that would automatically provide those users that do not have PKI certificates with either the certificates issued by the users' chat servers or with self-signed certificates.

CONCLUSION

Participation at RF07 was a valuable exercise for the MIT LL's Secure Dynamic Groups project. It allowed us to test our ideas and their prototype realizations in the real-use environment and to feed results of this testing into ongoing R&D, ultimately yielding solutions that are better tailored to the realities of tactical system usage and tactical networks. RF07 was a successful first step in preparing us for larger scale experimentation at other DoD exercises and future pilot deployments of GSC.

Acknowledgments: The team thanks the MIT LL 4xComm team, and especially Stephen McGarry, Randy Charland, Lenny Veytser, Mark Yaeger, Wayne Bynoe, Andrea Coyle, Orton Huang, and Igor Pedan, for helping us test GSC, providing us with useful feedback, and in general for their hospitability and fellowship at RF07.

MIT LL's RF07 participation, led by Stephen McGarry, was an extremely well managed and executed event – by being a part of this event we learned from and collaborated with the best.

REFERENCES

1. USAF Red Flag exercise, web resources: http://en.wikipedia.org/wiki/Red_Flag_(USAF), and http://www.globalsecurity.org/military/ops/red-flag.htm.
2. Red FLAG 07-3 Exercise Report Network Control Plane, Lincoln Laboratory Project Report AN-1, 2008.
3. Web resources: OpenFire server: www.jive.org; Pidgin client: www.pidgin.im; XMPP standard: www.xmpp.org.
4. D. Boneh, C. Gentry, and B. Waters, "Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys," Crypto, 2005.
5. R. Khazan et al., "Securing Communication of Dynamic Groups in Dynamic Network-Centric Environments", MILCOM, 2006.
6. R. Khazan et al., "Robust Collaborative Multicast Service for Airborne Command and Control Environment," MILCOM, 2004.