Proxychain: Developing a Robust and Efficient Authentication Infrastructure for Carrier-Scale VoIP Networks

Italo Dacosta and Patrick Traynor
Converging Infrastructure Security (CISEC) Laboratory
Georgia Tech Information Security Center (GTISC)
Georgia Institute of Technology
{*idacosta, traynor*}*@cc.gatech.edu*

Abstract

Authentication is an important mechanism for the reliable operation of any Voice over IP (VoIP) infrastructure. Digest authentication has become the most widely adopted VoIP authentication protocol due to its simple properties. However, even this lightweight protocol can have a significant impact on the performance and scalability of a VoIP infrastructure. In this paper, we present *Proxychain* – a novel VoIP authentication protocol based on a modified hash chain construction. Proxychain not only improves performance and scalability, but also offers additional security properties such as mutual authentication. Through experimental analysis we demonstrate an improvement of greater than 1700% of the maximum call throughput possible with Digest authentication in the same architecture. We show that the more efficient authentication mechanisms of Proxychain can be used to improve the overall security of a carrier-scale VoIP network.

1 Introduction

Voice over IP (VoIP) is fundamentally reshaping the telephony landscape. Instead of using dedicated, circuit-switched lines, VoIP allows for phone calls to be multiplexed with other data traffic over the Internet. This convergence between voice and data communications provides a number of benefits. For instance, providers can now offer a range of new services such as video and presence. Unfortunately, the transition from traditional phone networks to VoIP also creates a number of new security challenges.

Authentication represents one of the most important security issues facing VoIP systems. Providers have responded by implementing a number of security mechanisms, ranging from SSL/TLS to Digest authentication. Unfortunately, none of the suggested schemes are simultaneously strong, efficient *and* scalable enough to meet the needs of carrier-scale networks without vastly increasing the amount of deployed infrastructure.

In this paper, we develop *Proxychain*, a robust and efficient authentication infrastructure designed to support operations in carrier-scale VoIP networks. Our solution is built around a single centralized authentication service working with proxy nodes distributed across a wide geographic area. We reduce the impact of the latency and load associated with this architecture by using a modified hash chain construction (a sequence of one-time authentication tokens generated by applying a hash function repeatedly, once-per token, to a secret root value). In addition to providing an efficient mechanism for mutual authentication, our approach also provides improved scalability through the secure caching of temporary authentication tokens at the proxies. To the best of our knowledge, Proxychain is the first protocol that applies the idea of hash chains in the SIP domain. Proxychain not only adapts this idea to SIP authentication but also extends it by including additional modifications that solve some of the weaknesses associated with hash chain protocols, resulting in a more robust protocol.

This paper makes the following contributions:

- **Design and implementation of Proxychain:** We develop a construction based on modified hash chains. Our construction not only dramatically reduces the load on the centralized authentication database and the latencies associated with accessing it, but also provides mutual authentication for clients and providers.
- **Evaluation of Proxychain through an extensive measurement study:** We measure, characterize and compare the performance characteristics of our proposed infrastructure against commonly used mechanisms. Our results show up to a 1700% improvement over such schemes. Moreover, we demonstrate the ability to support the authentication needs of a national-scale VoIP network using unoptimized COTS hardware and databases.
- **Evidence of robustness to outages and downtime:**

We demonstrate that our construction allows the network to operate during planned and unplanned outages, and estimate its robustness to such incidents. We show the ability to support normal operations with high availability for approximately 6 hours using only 28 minutes of preemptive computation.

Improvements to the efficiency of SIP authentication afforded by Proxychain allow us to significantly increase the overall security of VoIP systems. For instance, several recently disclosed attacks on VoIP systems [2, 29] can be mitigated by simply having an authentication infrastructure scalable enough to cryptographically verify the origin of multiple SIP signaling request types (e.g., INVITE and BYE).

The remainder of our paper is organized as follows: Section 2 discusses a nation-wide VoIP architecture and provides important background information; Section 3 details our proposed protocol; Section 4 provides the details of our experimental setup; Section 5 shows the results of our experiments; Section 6 discusses a number of additional points; Section 7 presents related work; Section 8 offers concluding remarks and future work.

2 A Nationwide VoIP Infrastructure

Telephony networks have long relied on a series of distributed databases and proxies to implement authentication. However, advances in processor speeds and ease of management have prompted a number of cellular [18] and VoIP providers such as Skype to rely on a central authentication service.[1] Therefore, our authentication mechanism is designed to work for a similar architecture. Figure 1 shows our simulated testbed.

2.1 Session Initiation Protocol

The Session Initiation Protocol (SIP) [21] is the underlying architecture of the majority of VoIP systems. This application-layer signaling protocol has several components. End devices are known as *User Agents* (UA), and can act as a client (UAC) or as a server (UAS). UACs generate SIP requests, while the UAS generates responses to SIP requests. When attempting to establish a session with Bob, Alice sends her request to a SIP *Proxy* server. The proxy determines the IP address of Bob's UAS and forwards Alice's request. In addition to routing call setup requests, a proxy also participates in the process of authentication with the help of an authentication database.

SIP provides two message types: requests (client to server) and responses (server to client). There are six types of requests: *INVITE* (to establish a session between UAs), *ACK* (to acknowledge a reliable message exchange), *CANCEL* (to terminate a pending request), *BYE* (to terminate a existent session), *OPTIONS* (to query for the capabilities of servers), and *REGISTER* (used by a UA to notify its current IP address to a Registrar process running on the proxy). The responses are grouped into six categories and indicate the status of a current request. For example, a *200 OK* response indicates a successful transaction (more detail in the RFC 3261 [21]).

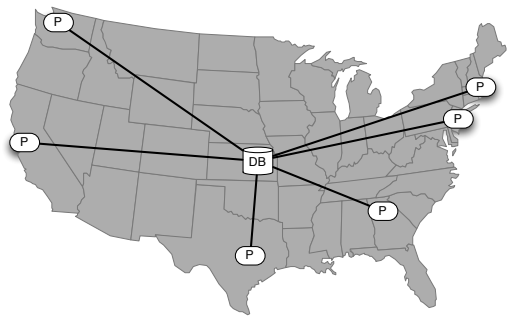


Figure 1: The hypothetical nationwide SIP infrastructure modeled in our experiments using latencies collected from Planetlab. As it is done by some cellular providers, our authentication service (DB) is centrally located, with proxies (P) distributed across the country.

2.2 Digest Authentication

SIP Digest authentication is a challenge-response authentication protocol based on HTTP Digest authentication [11]. Digest authentication is used by SIP proxies to validate the identity of requests received from UAs. It allows users to prove their knowledge of a shared secret (e.g., password) to a server without sending the secret unprotected over the network (protection against eavesdropping attacks). Digest authentication is widely supported because it is more efficient and easier to implement than the other protocols recommended by RFC 3261 (i.e., TLS, S/MIME, IPsec). Furthermore, it is the only authentication protocol required in the UAs according to RFC 3261 (support for other protocols is not required).

Figure 2 shows a SIP call dialog using Digest authentication. As in most deployments, only INVITE requests require authentication. First, Alice's UA sends an INVITE request to the proxy. The proxy determines that the request requires authentication and responds with a SIP 407 response ("`Proxy Authentication Required`") containing a nonce. Alice's UA acknowledges the reception of the challenge, computes the hash of the shared secret and the nonce and sends it back to the proxy using a new INVITE message. The proxy then computes the answer after querying a database that stores the user's shared secret. Finally, the proxy compares both values and, if they match, forwards the INVITE to the destination and the SIP dialog continues its standard flow.

Digest authentication efficiency relies on the use of hash operations and nonces, instead of symmetric or public key cryptography. In its basic form, a Digest authenti-

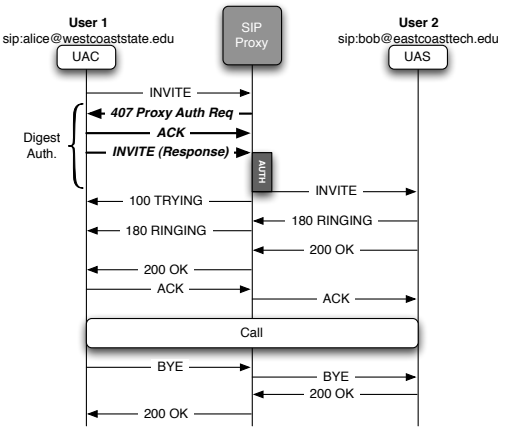


Figure 2: SIP call setup using Digest authentication (bold).

cation response is computed as follows:

Response = H(H(uid||realm||pwd) ||n|| H(method||URI))

where $H()$ is a cryptographic hash function (MD5 is the default), $||$ corresponds to a concatenation operation, *uid* is the user ID, *realm* is the proxy's protection domain, *pwd* is the user password, n is a nonce, *method* is the SIP request authenticated (e.g., INVITE) and *URI* is the destination address Alice is trying to reach (e.g., bob@eastcoaststate.edu).

2.3 Problems with Digest Authentication

While more efficient, Digest authentication is less secure than protocols such as TLS or IPsec. For instance, it does not provide mutual authentication and complete message integrity. Limited integrity protection is offered but it is optional and not widely supported by UAs. Additionally, current implementations actually send the shared secret from the database to the proxy in order to calculate the correct client response. This approach is dangerous if the proxy is compromised. Several vulnerabilities have been published regarding commercial SIP deployments due to these weaknesses [29].

The use of Digest authentication in an environment with a remote authentication service dramatically reduces performance. The main reason is that authentication operations become more expensive - the round-trip time (RTT) between a proxy and the database (tens of milliseconds) is now added to each authentication operation (hundreds of microseconds). The additional time added per call setup reduces the call throughput of each proxy. The problem is exacerbated by the fact that proxies have to query the database for each SIP message that requires authentication. This action also creates a considerable network load when the call throughput is high. If multiple proxies are used, the load could overwhelm the database or its network link. As a result, scalability is also affected.

The use of multiple databases (i.e., one local database per proxy) or adding more hardware resources to the database are not efficient solutions. Dacosta et al. [8] showed that the effects of network latency could be reduced by a combination of parallelization and batching techniques. However, the network load to the database is still high enough to affect the scalability of the system. A more efficient approach is to reduce the number of queries to the database. To achieve this, we can use temporary authentication credentials that each proxy stores in memory and that can be used in multiple authentication operations without contacting the database. This approach reduces the load received by the database and the effects of network latency. Our proposed protocol follows this approach.

3 Proxychain Protocol Specification

3.1 Hash Chains

A hash chain is created by applying a cryptographic hash function $H()$ (e.g., MD5, SHA-1) multiple times to a random value r to generate a sequence of values that can be used as one-time authentication tokens. A hash chain of length n is computed as:

$$H^n(r) = H(\ldots H(H(r))\ldots)$$

Hash chains rely on the preimage resistant (i.e., one-way) property of cryptographic hash functions. When attempting to authenticate to a server possessing $H^n(r)$, the client transmits $H^{n-1}(r)$. The server then hashes $H^{n-1}(r)$ a single time and, if the result matches $H^n(r)$, authenticates C based on the computational infeasibility of an adversary guessing the correct preimage.

3.2 Design Goals

Proxychain design addresses some of the shortcomings of Digest authentication in SIP topologies with a centralized authentication service. Our first goal is *efficiency:* Proxychain should execute authentication operations faster than Digest authentication, allowing improved call throughput. Second, we focus on *scalability:* Proxychain should support more users and proxies than Digest authentication without the need for additional resources. In particular, Proxychain should reduce the bandwidth and processing time required by the database to avoid bottlenecks. Finally, our third goal is *security:* Proxychain should improve upon the security assurances provided by Digest authentication.

3.3 Design and Formal Description

Proxychain is designed to reduce the impact of latency and load on the remote authentication service by caching

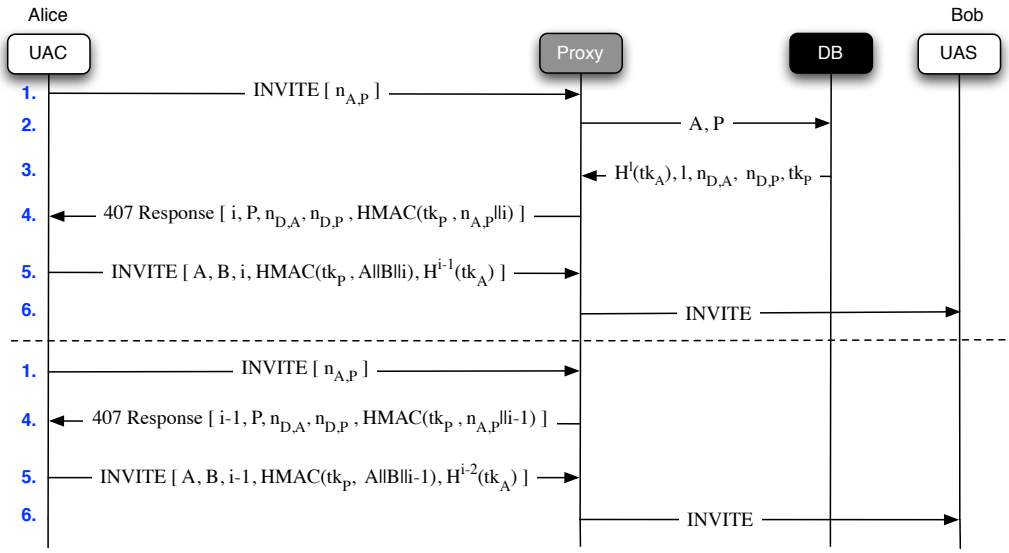


Figure 3: Call setup flow using Proxychain. For the first request (above dashed line), the proxy must request a temporary credential from the database. Subsequent requests (below dotted line) can be dealt with immediately by the proxy.

1. $A \rightarrow P : A, B, n_{A,P}$
2. $P \rightarrow D : A, P$
3. $D \rightarrow P : H^l(tk_A), l, n_{D,A}, n_{D,P}, tk_P$
4. $P \rightarrow A : i, P, n_{D,A}, n_{D,P}, HMAC(tk_P, n_{A,P}||i)$
5. $A \rightarrow P : A, B, i, HMAC(tk_P, A||B||i), H^{i-1}(tk_A)$
6. $P \rightarrow B$

A, B, P, D : Alice, Bob, Proxy, Database
$k_{A,D}$: Secret key between Alice and database
$n_{D,A}, n_{D,P}, n_{A,P}$: Nonces
l : Hash chain length
i : Hash chain current sequence number
$H^i(x)$: *i-th* hash value of *x*, H(H(...H(*x*)...))
$\text{HMAC}(k, x)$: HMAC with key *k* on *x*
tk_A : $\text{HMAC}(k_{A,D}, n_{D,A}||P)$
tk_P : $\text{HMAC}(k_{A,D}, n_{D,P}||P)$

Figure 4: *Proxychain protocol:* The formal definition of the Proxychain protocol. We assume that there exists an encrypted channel (e.g., IPsec connection) between the proxy and the database.

temporary authentication credentials at the proxies. Using hash chain-based credentials of length l, a proxy can authenticate multiple requests from a particular user with only $\frac{1}{l}$ queries to the database. The database creates credentials based on the secret it shares with each user and determines the credential's parameters, including length, hash function, and expiration time. This approach is more secure than the associated Digest authentication mechanism, as the shared secret between the database and the user is never exposed to the proxies. A compromise of one of these servers, therefore, does not necessarily require password resets for large number of users.

Each proxy provides services only to users that are geographically close to it (i.e., based on IP address or ZIP code information), much like a traditional telephony switch. Each proxy accordingly needs to store credentials for only a subset of the total number of users in the system. We explore the overhead associated with such credential storage in Section 4.2.

Figures 3 and 4 provide graphical and formal definitions of the Proxychain protocol, respectively. A user Alice attempts to call Bob by first sending an INVITE request to her proxy, which contains the source and destination of the call and a nonce $n_{A,P}$ (Message 1). The proxy checks to see if it has a credential for Alice and, if not, queries the authentication database with the identifiers corresponding to Alice and the proxy (A, P) for a new hash chain (Message 2). Note that requests between proxies and the authentication database occur over a long-lived, encrypted and authenticated channel such as IPsec

or TLS/SSL. The database generates a five-tuple that includes a new hash chain ($H^l(tk_A)$), the length of the hash chain l, nonces for both the proxy and Alice ($n_{D,P}$ and $n_{D,A}$), and a session key tk_P. The hash chain is calculated as $H^l(\text{HMAC}(k_{A,D}, n_{D,A}||P))$, and the session key as $\text{HMAC}(k_{A,D}, n_{D,P}||P)$ (Message 3).

After receiving the tuple from the authentication database, the proxy returns a `407 Proxy Authentication Required` SIP message to Alice. This message includes a counter $i \leq l-1$, the proxy's identifier P, the two nonces generated by the authentication database ($n_{D,P}$ and $n_{D,A}$) and a network authentication token $\text{HMAC}(tk_p, n_{A,P}||i)$ (Message 4). The client receives the response and uses $k_{A,D}$ to calculate the session key tk_P and then authenticates the message from the proxy. If the message authenticates properly, Alice then generates her session key tk_A and hashes it $i-1$ times to generate $H^{i-1}(tk_A)$. Alice responds to the proxy by sending a new INVITE message containing A, B, i, $HMAC(tk_P, A||B||i)$ and $H^{i-1}(tk_A)$, which the proxy hashes forward a single time (assuming that the HMAC properly verifies) (Message 5). If $H(H^{i-1}(tk_A)) = H^i(tk_A)$, then the proxy records $H^{i-1}(tk_A)$ as the next legitimate credential, decrements i and the INVITE request is forwarded to Bob (message 6). On subsequent authentication attempts by Alice where $c < i-1$, the proxy responds to Message 1 with Message 4, which contains $c, P, n_{D,A}, n_{D,P}, HMAC(tk_P, n_{A,P}||c)$.

Note that unlike Digest authentication, Proxychain provides mutual authentication. Specifically, the network authentication token $HMAC(tk_p, n_{A,P}||i)$ can only be produced with knowledge of tk_P and using the nonce supplied by the user Alice. Moreover, because only the user and the authentication database could have created tk_P (because only they have knowledge of $k_{A,D}$), an adversary can not create legitimate hash chains without the assistance of the authentication database.

4 Experimental Setup

4.1 Experimental Testbed

Our experimental testbed is based on the VoIP infrastructure depicted in Figure 1. As this figure shows, the testbed is composed of three main components: the authentication database, SIP proxies and the user clients (UAs). The database and proxies are run on servers from the Georgia Tech Emulab testbed.[2] We use seven servers to represent the infrastructure (one database and six proxies). These servers run Linux Kernel 2.6.26 (Fedora Release 8), have two 2.80 GHz Intel Xeon processors and 512 MB of memory. The UAs are run on servers from our research lab. A total of nine servers are used, each running multiple UA instances to generate call traffic. These servers run Linux Kernel 2.6.24 (Ubuntu 8.04.2), eight (8) 2.00 GHz Quad-Core AMD Opteron processors and 16 GB of memory.

The network latency between the proxies and the database is simulated using Emulab's traffic shaping functionality. In order to use realistic latency values, we performed measurements using the Planetlab network testbed.[3] Using the ping network tool, we measured the round-trip time (RTT) between a Planetlab node located in the University of Kansas and Planetlab nodes located at UC Berkeley (67.6 ms), Georgia Tech (33.1 ms), MIT (44.7 ms), Princeton (43.8 ms), the University of Texas (20.6 ms), and the University of Washington (43.4 ms). The RTT data was collected during a 24 hours period and average values were calculated. Finally, no additional latency values were simulated between the proxies and the UAs (latency was around 1 ms). The reason is that our testbed assumes physical proximity and low latency values (e.g., $<$ 10 ms) between the UAs and the proxies. Simulating this latency is not necessary because it would not affect the test load generated by the UAs and our results (it would slightly affect the setup time of each call).

The proxies are implemented using OpenSIPS[4] 1.5.2. OpenSIPS is a mature open source SIP proxy optimized for high performance. The proxies are configured with minimal functionality (stateless configuration and basic modules required for routing). We run MySQL[5] 5.0.45 as our database, a well-known open source relational database management system. MySQL is run with a default configuration (no optimizations). Finally, SIPp[6] 3.1 is used to generate the UAs' workload, which conforms to a uniform random distribution. SIPp is an open source traffic generator for the SIP protocol. A total of 36 SIPp instances are used in our experiments (18 UACs and UASs). Default SIPp scenarios are modified to support INVITE and BYE authentication for Digest and Proxychain authentication (SIP call flows in Figures 2 and 3).

Each proxy serves requests for 200,000 unique users. The number of users per proxy is limited by the proxy's available memory, disk space in the database and the size of authentication credentials (see Section 4.2). As a result, the total number of users in the database is 1,200,000. All the users are part of a single SIP domain (no inter-domain calls).

4.2 Proxychain Implementation

Implementing Proxychain requires a combination of new code modules and modifications to existing software. In the proxies, OpenSIPS ($\approx$ 320000 lines of code (loc)) required approximately 710 loc to support Proxychain. In the UAs, SIPp ($\approx$ 3000 loc) required around 140 loc. In the database, we built a separate concurrent-process server application to handle queries from proxies and the associated cryptographic operations. This server application required approximately 880 loc. The

MySQL database software itself was unmodified. All of our experimental code, which was written in C, and supporting scripts are available at `http://www.cc.gatech.edu/~idacosta/proxychain.html`

Proxychain uses the same SIP headers in the challenge and response messages. For example, a Proxy-Authenticate header (challenge) looks as follows:

```
Proxy-Authenticate:PC realm="CISEC", i="10",
nda="0ec497d9a5ba5e1f2b2177d83fb3d341",
ndp="f1e992583dd5daecddea3309a01e5347",
hmac="15f5d33206e79eaea7245682d9953164"
```

where *PC* indicates the use of the Proxychain protocol, *realm* is proxy's identifier, *i* is the sequence number, *nda* and *ndp* are the nonces and *hmac* is the network authentication token.

The corresponding Proxy-Authorization header using Proxychain looks as follows:

```
Proxy-Authorization:  PC username="0000001",
realm="CISEC", i="10",
response="a0843d4b8a712284ff5a6fcd136c4b47,"
hmac="f9fd4ef6689850406a560965a4381c57"
```

where *response* is the next value in the hash chain sequence. The other parameters have the same meaning as in the Proxy-Authenticate header.

Our Proxychain implementation uses the MD5 hash function in order to compare it more directly and fairly to Digest authentication. Nevertheless, our code requires few modifications to support SHA-1. With MD5, the size of a temporary authentication credential is 134 bytes. As a result, each proxy in our testbed requires a minimum of 26 MB of free memory for serving 200,000 users.

4.3 Methodology

We perform a number of different experiments in order to characterize Proxychain. We specifically compare our protocol against a system with no authentication mechanism and one using Digest authentication. We do not measure more computationally expensive mechanisms such as TLS/SSL as previous studies have demonstrated that they provide significantly lower throughput [5, 6, 15, 16]. We collect the following metrics in most of our experiments: call throughput, message retransmissions, failed calls, bandwidth utilization and database CPU utilization. These are global metrics, the totals for the whole infrastructure (i.e., the call throughput is equal to the sum of the call throughput measured in each proxy).

The *call throughput* refers to the number of successful calls per second (cps) measured every five seconds.

Message retransmissions corresponds to the number of SIP messages retransmitted due to the expiration of timers in the UAs. Our tests use the default retransmission time defined by SIP standards (500 ms). *Failed calls* refer to the total number of unsuccessful calls measured in the last period. In our experiments, we consider only calls failures due to maximum number of retransmissions (maximum number of UDP retransmissions attempts has been reached). We use the default values in SIPp for the maximum number of retransmissions: five for INVITE messages and seven for others. Finally, *bandwidth utilization* corresponds to the total network throughput (KBytes/sec) measured from the database during each test.

During our experimental analysis, each test was run at least 10 times to ensure the soundness of the results. Average values are used in our analysis and a 95% confidence interval is provided in most of the graphs. Note that these bounds are often difficult to observe in our graphs as the values are generally very close to the mean.

Protocol	Digest	Stdev	Proxychain	Stdev
Response (μsec)	116.81	13.59	184.76	49.92
Verification (μsec)	197.24	21.51	66.97	15.07

Table 1: Response computation time at the UA and verification time at the proxy for Digest and Proxychain authentication. Proxychain adds little overhead to the response computation and it is more efficient performing verifications.

Length	10	100	1000	10000
Time (μsec)	294.10	335.15	1383.53	11875.71
Stdev (μsec)	18.42	15.28	18.07	120.44

Table 2: Time required by the database to compute credentials with different hash chain lengths. For lengths < 100, the overhead is small.

5 Experimental Results

5.1 Microbenchmarks

To understand the computational differences between Digest and Proxychain authentication, we measure the time to compute a response in the UA and the time to verify a response in the proxy. To measure these values, we use network traces (100 samples per value). For Proxychain, the measurements are performed the first time a credential is used (hash chain length of 10). This corresponds to the worst case for response computation because it requires the highest number of hash operations (9 operations).

Table 1 shows the results. The UA running Proxychain requires approximately 70 μsec of additional computation than one running Digest authentication. This difference is due to the additional integrity checks and hash operations required by Proxychain in the UA. However, this difference is not significant as individual UAs does not perform large amounts of computation in this system. Interestingly, the response verification is nearly three times faster when Proxychain is used by the proxy. The rea-

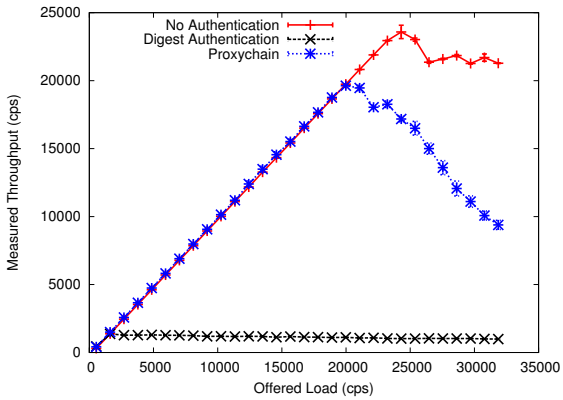


Figure 5: Total call throughput for no, Digest and Proxychain authentication. Proxychain's maximum call throughput is close to the one obtained without authentication.

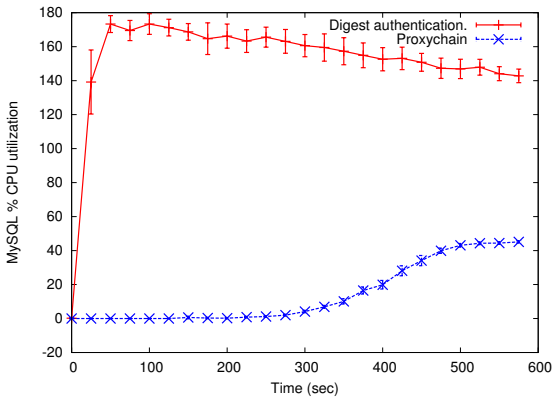


Figure 6: Percentage of CPU required by the database process for Digest and Proxychain authentication. The database process is virtually idle when Proxychain is used.

son is that Proxychain only requires two hash operations to verify a response. On the contrary, Digest authentication requires three hash operations and additional checks to verify a response. Based on these results, we argue that the computational overhead added by Proxychain is not significantly different from the one added by Digest authentication.

We also evaluate the overhead of generating hash chains of varying lengths. Specifically, we measure the time required by the authentication database to generate credentials of lengths 10, 100, 1000 and 10000. As before, we use network traces to measure the time for each configuration (100 samples per configuration). Table 2 shows the results of these experiments. As expected, increasing the hash chain size increases the time required to generate credentials. The additional time remains small for hash chains with length up to 100 (< 350 μsec).

5.2 Call Throughput

Microbenchmarks provide insight into the overhead that can be expected at each component of the network. However, they do not provide a picture of the overall behavior of a system. Accordingly, we characterize the interaction of those components by measuring total call throughput. We compare throughput for systems configured to use Digest authentication, Proxychain and no authentication mechanism. UAs generate an increasing call load (270 cps increments every 5 seconds) over the course of 10 minutes. In addition, we evaluate the best configuration for each protocol. For Digest authentication, we use close to 100 concurrent proxy-processes per proxy.[7] For Proxychain, we preload each proxy with all its user credentials (200K credentials with hash chain length of 10) before each experiment and use 8 concurrent proxy-processes per proxy (OpenSIPS recommended value).

Figure 5 shows the results of these experiments. Without authentication (baseline configuration), the network supports a maximum call throughput of nearly 24,000 cps. When Digest authentication is used, the maximum call throughput drops dramatically to approximately 1,160 cps. This result represents a 95% reduction in call throughput when compared with the baseline configuration. For Proxychain, the result is more favorable: a total call throughput of over 19,700 cps. In this case, the call throughput drops by only 18% when compared with the baseline configuration. However, when compared to Digest authentication, Proxychain allows an increase of over 1,700% (more than an order of magnitude). Accordingly, Proxychain is significantly more efficient than Digest authentication in this architecture.

Figure 6 provides insight into the poor performance of Digest authentication. The database process rapidly reaches 175% CPU utilization (dual-core machine). This behavior indicates that queries from the proxies saturate the authentication database, making it a bottleneck. We observe the opposite when using Proxychain. The database was virtually idle ($< 5\%$ CPU utilization) before the system reaches its maximum call throughput, at which point the system becomes unstable due to the high number of retransmissions.

A naïve solution to improve Digest authentication performance would be to use a more powerful database. Therefore, we repeated the experiment using a quad-processor server for the database. As expected, the maximum call throughput increases, but only to approximately 4,000 cps. However, in this experiment the database does not saturate - CPU utilization is below 300%. In this case, throughput fails to increase further due to the network latency between the proxies and the database.

Another important difference is the total bandwidth required for both configurations. The message overhead between a UA and the proxy are arguably equivalent. Message 4, the challenge, requires an additional 92 B and 165 B for Digest and Proxychain authentication, respectively. The response in Message 5 similarly requires an addi-

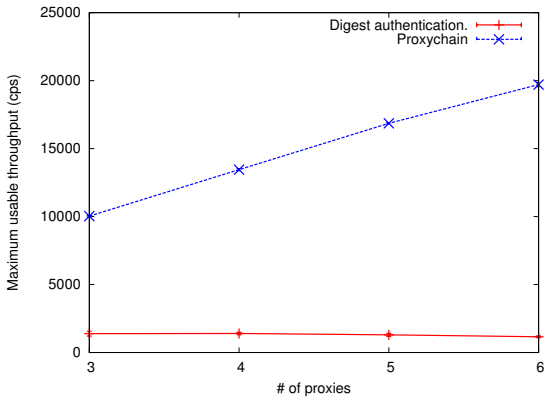


Figure 7: Throughput measured for a range of proxies using Digest and Proxychain authentication. Proxychain is considerably more scalable than Digest authentication.

tional 199 B and 153 B. At its maximum call throughput (measured from the database), Digest authentication required almost 130 and 430 KBytes/sec for queries and responses respectively. In contrast, Proxychain required less than 1 KByte/sec for both, queries and responses. As expected, the use of temporary credentials significantly reduces the total number of queries to the database.

The previous results also mean that increasing the hash chain length (>10) will not help to improve performance in our testbed. The reason is that the load in the database is already low with a hash chain length of 10. Using a longer size will make the load even lower but the difference will not affect the overall performance of the system. On the contrary, using hash chains that are too long could affect performance because of the additional hash operations that will be needed by the UACs and the database.

Finally, for the baseline and Proxychain configurations, the maximum call throughput is limited by the proxy application itself: OpenSIPS. Analyzing the resources usage statistics (memory, CPU and bandwidth) collected during the experiments for the different testbed components, we find that none of the resources are completely used (no shortage of resources) when the two configurations reach the maximum call throughput. Based on this evidence and in our experience with OpenSIPS, we can conclude that the OpenSIPS software is the performance bottleneck for no authentication and Proxychain configurations. Using an optimized version of OpenSIPS or a faster proxy server application will provide higher call throughput values.

5.3 Scalability

In this set of experiments, we evaluate how the testbed handles an increasing number of users, and therefore, an increasing load. To simulate a varying number of users, we measure performance with a varying number of proxies, where each proxy represents 200,000 users. Using a similar procedure as in the previous test, we measure the call throughput for 3, 4, 5 and 6 proxy configurations (600K, 800K, 1M and 1.2M users respectively).

The results are presented in Figure 7. We can see that for Digest authentication, the maximum call throughput measured is approximately the same (≈1,200 cps; linear regression: $y = -79.6x + 1670.5$ $R^2 = 0.848$ [8]) for all the configurations. The reason is that even for a three-proxy configuration, the database becomes saturated rapidly (see previous test). Therefore, Digest authentication limits the scalability of the system. For Proxychain, the maximum call throughput increases linearly with the number of proxies (≈3,250 cps per proxy; linear regression: $y = 3243.9x + 416.5$ $R^2 = 0.998$). From these results, we can conclude that Proxychain allows the system to grow by just adding new proxies and without requiring changes to the database.

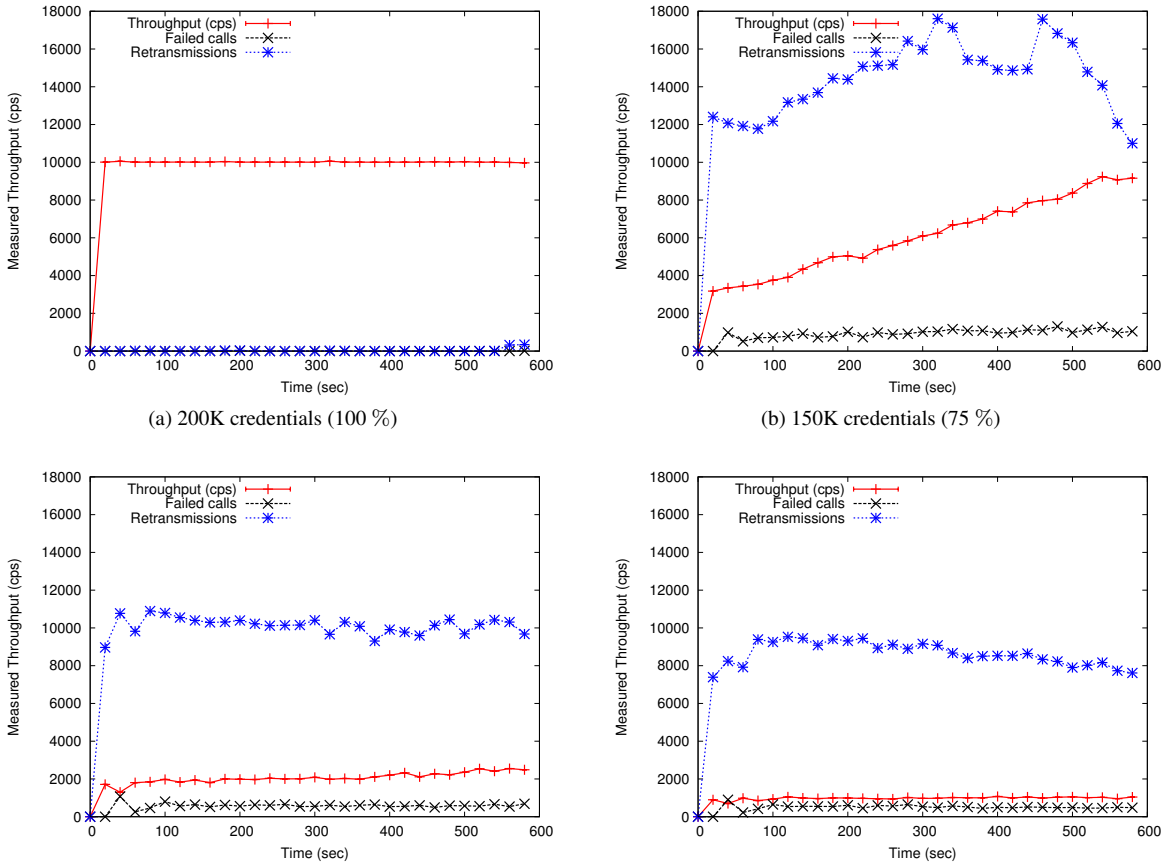
5.4 Credential Preloading in the Proxies

In the previous tests, we evaluated Proxychain's performance using a best-case scenario: each proxy had all the credentials in memory before the tests started. We now evaluate performance when a lower number of credentials are preloaded in each proxy. For this purpose, we use a similar procedure as in previous tests but with two exceptions. First, we use a constant workload of 10,000 cps with no ramp-up period. Second, we preload the proxies with 200K, 150K, 100K and 50K credentials in each test.

Figure 8 shows the results for all the configurations. For the 200K configuration (best-case, Figure 8a), the call throughput reaches 10,000 cps quickly (< 10 sec) with virtually no message retransmissions or failed calls. For the 150K configuration (Figure 8b), the call throughput jumps to approximately 3,000 cps, and then continues increasing until it reaches almost 10,000 cps by the end of the test. However, a large number of retransmissions and failed calls occur. Finally, for the other two configurations (Figures 8c and 8d), the behavior is worse. The maximum call throughput measured was around 2,000 and 1,000 cps respectively during the experiments. The number of retransmissions and failed calls is also constantly high. In theory, each configuration should have reached 10,000 cps after some period of time. However, the large number of retransmissions makes the system unstable. These results show the importance of having most of the credentials stored in the proxies to avoid the negative effects of retransmissions, especially when high loads are expected.

5.5 Prefetching mechanism

The previous test shows that Proxychain is more effective if each proxy has credentials for almost all its users (best case scenario). However, credentials are stored or updated in the proxy only after a user request that requires authentication. Therefore, we implement a prefetching mecha-



(a) 200K credentials (100 %)

(b) 150K credentials (75 %)

(c) 100K credentials (50 %)

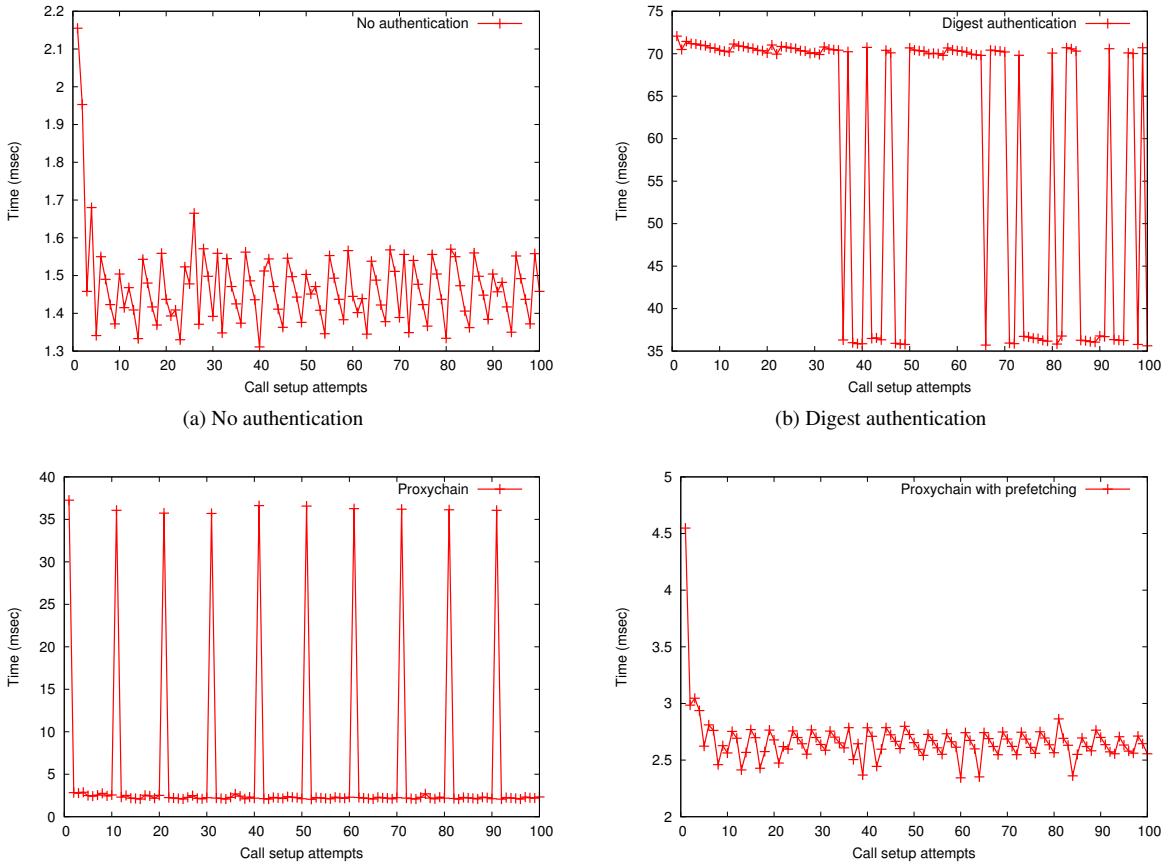
(d) 50K credentials (25 %)

Figure 8: Call throughput measured for different number of credentials preloaded in the proxies and a constant offered load (10K cps). Proxychain requires that proxies have most of the credentials in memory for maximum performance.

nism that automatically queries the database for credentials without requiring any user action. This mechanism, running as a separate proxy process, checks if a user has a credential in the proxy or if her credential has already expired (i.e., $l = 0$). In short, the prefetching mechanism guarantees the best case scenario for Proxychain.

In this experiment, we characterize the effect of the prefetching mechanism on the call setup time for individual UAs (time elapsed between the first INVITE request and the 200 OK response). We use a UA sending a low load (< 5 cps) to a single proxy and estimate the call setup time using network traces (100 samples). Four proxy configurations are used: no authentication, Digest authentication, Proxychain and Proxychain with prefetching.

Figure 9 shows the results for each configuration. As expected, when no authentication is used (Figure 9a), the call setup is the fastest: 1.47 ms on average. For Digest authentication (Figure 9b), we can observe the effects of the RTT between the proxy and the database ($\approx$33 ms) on the call setup time. Two call setup times are measured: 36 and 71 ms approximately. The reason is that for the first value, only one RTT is required during call setup, while for the second value, two RTTs are required due to the low test load used (no TCP piggybacking). In general, only one RTT is required, so we can assume that the call setup time for Digest authentication is approximately 36 ms. In the case of Proxychain, Figure 9c shows how the temporary credentials reduced the call setup time while they are valid. While the credentials are active (hash chain size $>$ 0), the call setup time is only 2.27 ms on average. Once a credential expires (hash chain size = 0), a query to the database is required, so the call setup time increased by one RTT: 36.28 ms on average. When Proxychain is used with prefetching (Figure 9d), the average call setup time is only 2.67 ms. The reason is that no credential updates are performed during call setups. Instead, credentials are updated by the prefetching process automatically, before they are required in a call setup. Therefore, the call setup time when Proxychain is used with prefetching is close to the call setup time when no authentication is used ($\approx$1 ms difference). Accordingly, prefetching helps to eliminate the effect of network latency on call setup time.



(a) No authentication

(b) Digest authentication

(c) Proxychain

(d) Proxychain with prefetching

Figure 9: Call setup time for four different configurations: no, Digest, Proxychain and Proxychain with prefetching authentication. The call setup time for Proxychain with prefetching is similar to the one obtained with no authentication.

5.6 Authenticating Multiple Message Types

In our final set of experiments, we explore the effect of authenticating multiple SIP message types request per session (call dialog). For example, the lack of authentication of BYE requests allows several reported attacks against SIP deployments [29]. However, if BYE requests are also authenticated using Digest authentication, the performance of the system will decrease even more due to the additional operations and queries to the database. In this experiment, we evaluate the impact of authenticating INVITE and BYE requests on performance when Proxychain is used. We use a similar procedure as in Section 5.2 (i.e., no prefetching). The only difference is that the proxies and UACs are configured to authenticate BYE in addition to INVITE requests.

Figure 10 shows the call throughput for the two configurations: INVITE and "INVITE and BYE" Proxychain authentication. As expected, the maximum call throughput supported by the testbed decreases when two requests are authenticated to approximately 12,000 cps. This represents a performance drop of nearly 50%. The reason is that credentials are used faster (twice as fast) because two authentication operations are required per call, making the number of queries to the database increase, resulting in higher CPU and bandwidth utilization. However, the use of Proxychain to authenticate two types of signaling messages still provides over 800% greater throughput than Digest authentication authenticating a single message.

Finally, we test if increasing the hash chain length improves the performance in this scenario. The idea is that, if credentials are used faster when two requests per call are authenticated, increasing the hash chain length should reduce how fast they need to be replaced. This will result in lower load to the database and increased throughput. The experiment confirms our hypothesis: using a hash chain length of 20 results in a maximum call throughput of almost 14,000 cps. This represents an improvement of almost 17% when compared to using hash chain length of 10. However, increasing the hash chain length further does not improve performance. On the contrary, the performance drops back to almost 12,000 cps with a hash chain size of 30 (using a longer hash chain caused ear-

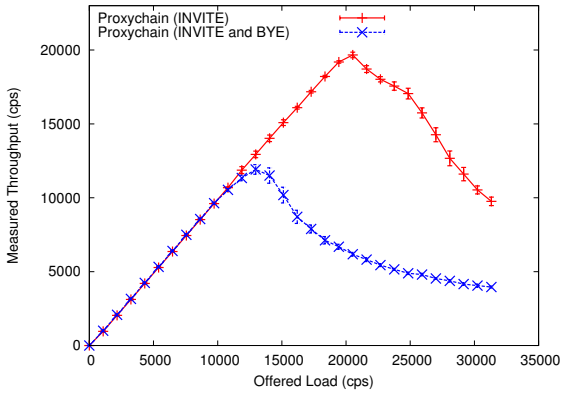


Figure 10: Throughput for INVITE and INVITE and BYE Proxychain authentication. Proxychain allows authentication of two requests per call while still supporting high throughput.

lier retransmissions, which affects the performance). The reason for these results is that we again reach the limits of the proxy application. The call throughput achieved is lower because the authentication of two requests involves additional messages and operations.

6 Discussion

6.1 Performance

The results presented in the previous section show that Proxychain effectively addresses the limitations of Digest authentication in VoIP topologies with a centralized authentication service. Specifically, Proxychain reduces the effects of network latency, allowing higher throughput. In our testbed, Proxychain's performance improvement was enough to reach the limits imposed by the proxy application (OpenSIPS). Moreover, Proxychain reduces the load received by the database, improving scalability.

The caching of temporary authentication credentials across the proxies allows our solution to perform so much better than Digest authentication. Not surprisingly, cellular networks perform a similar distributed caching of credentials, which are generated by a *Home Location Register* (HLR) and stored in the Mobile Switching Center/Visitor Location Register (MSC/VLR) closest to the client. However, the Proxychain approach is more efficient in terms of memory. Specifically, the current approach used in cellular networks requires that multiple credentials are stored in the MSC/VLR per user. Should the authentication database (HLR) wish to reduce its load, the proxies (MSC/VLRs) would need to be equipped with additional memory. Because Proxychain authentication credentials require a *constant* amount of memory regardless of the hash chain length, our approach is also more scalable than traditional caching. This property is particularly advantageous as it allows for more dynamic behavior by the infrastructure. For example, a database could monitor the received load and automatically increase the length of the hash chains in response to a spike in the load (e.g., busy hours, DoS attack or a flash crowd). We plan to explore such dynamic reprovisioning in future work.

The performance gains obtained in our experiments are based on the assumption that each proxy has most of its users' credentials most of the time. We also assumed that each proxy has a fixed set of registered users and that users do not register with other proxies often (e.g., traveling to another state). These assumptions can be relaxed by providing additional cache space in the proxies. For example, each proxy will have a cache of fixed size, and keep in the cache the credentials of the most active users. When new users register with a proxy, the proxy can use an eviction policy to replace the credentials in the cache based on frequency of use. In this way, each proxy could handle a variable number of users (more flexibility). This approach will be evaluated in future work.

The call throughput numbers achieved in our testbed could be considered high for commercial VoIP deployments. For example, AT&T average nationwide call volume is estimated to be around 300M calls per day, or an average of 3,472 cps [10], or roughly 17% of the throughput provided by our architecture. We note that while our testbed lacks some of the other functionality that a provider may chose to deploy (e.g., billing, media gateways), the performance benefits provided by Proxychain represent a significant potential improvement to real networks. Specifically, the additional capacity offered by Proxychain can serve as a defense mechanism to handle unexpected increments of requests for service.

The performance gains obtained by Proxychain requires some trade-offs. First, a proxy using Proxychain requires to keep a small amount of state for all its users (credentials), which is not necessary for Digest authentication. However, our experiments demonstrated that this was not a significant burden. UACs also need to perform more authentication operations when Proxychain is used. Specifically, Proxychain requires additional integrity checks and hash chain computations required to create a response. Nevertheless, the most expensive operations are hash computations that are in general very efficient to execute. In addition, the use of adequate hash chain lengths (i.e., < 100) and caching intermediate results in the UAC can reduce these overheads. Third, the database also requires to perform computation to create the user credentials. However, this is a one-time cost and it is lower than processing an equivalent number of requests per user as in Digest authentication.

In general, any SIP infrastructure with multiple proxies and a remote central authentication service will benefit from Proxychain, even if the performance requirements are not carrier-level. For example, the SIP infrastructure of a multinational corporation where each regional office

has a SIP proxy and the central database is located in the headquarters. The use of Proxychain in this scenario will reduce the load to the database (lower bandwidth and CPU utilization) and provide more security. As our results shows, the main requirement is to cache the credentials of most of the users (e.g., $> 75\%$) served by each proxy. This is not a hard requirement given the size of the credentials and the memory costs. Even in environments with high mobility requirements, caching the credentials of all the users in all the proxies or using caching algorithms are reasonable options. Finally, the concepts behind Proxychain can also be used in other domains with similar topology requirements. For example, remote authentication services such as RADIUS or DIAMETER, or authentication in IP Multimedia Subsystem (IMS) deployments could benefit from the performance, scalability and security advantages offered by Proxychain.

6.2 Security and Threat Analysis

Proxychain not only offers the security advantages of hash chains protocols (i.e., protection against eavesdropping and replay attacks), but also solves some of the weaknesses associated with these protocols [17]. For example, Proxychain provides integrity protection of the challenge in the form of an HMAC. This feature protects against an attacker located between the proxy and the client trying to change the counter (i) in the challenge to a lower value ($i = 1$) to obtain the complete hash chain sequence (small *n* attack [12]). Also, Proxychain provides mutual authentication between the UA and the proxy through the use of the session key generation based on a shared secret. The mutual authentication provided by Proxychain is less expensive and easier to implement than the one provided by protocols such as TLS or IPsec. In addition, Proxychain does not require hash chain synchronization[9] as S/Key does. The reason is that the hash chains are generated based on a secret derived from users' passwords.

Proxychain's threat model assumes that the database has a high level of security (a central database model facilitates this assumption). Only trusted entities (i.e., proxies) are allowed to communicate with the database using a robust security protocol (e.g., TLS or IPsec). Therefore, threats against the database can be considered low risks. In contrast, the proxies and the network traffic between proxies and UAs have a higher risk of being targeted by both, active and passive attackers.

An active attacker could try to compromise a proxy and steal its cached credentials. However, *Proxychain credentials cannot be used to impersonate users* (another advantage of hash chains). Instead, stolen Proxychain credentials could only be used to impersonate the proxy to the users due to the session key included in the credentials. In this scenario, only mutual authentication will be affected, resulting in the same security level provided by Digest authentication or S/Key (no server authentication). Therefore, an attacker will still need considerable effort to impersonate users even if she manages to steal the credentials cached by the proxy. While not implemented in our testbed, Proxychain can also include a revocation mechanism where the database can invalidate the credentials cached in a proxy. This mechanism will be useful in situations where a user needs to change her password or when a proxy has been compromised.

In addition, an active attacker could try to resend a previously captured user's response to make unauthorized requests in behalf of the user (replay attack). Proxychain provides a stronger defense against this type of attacks than Digest authentication due to the one-time password property of hash chains. Even if the attacker manages to capture a valid user's response (e.g., a MITM attack where the attacker prevents the user's response to reach the proxy), she will not be able to use it arbitrarily. The reason is that a Proxychain response includes the origin and destination of the request which are verified by the proxy (the attacker cannot modify the response because its integrity is protected).

Passive attackers (e.g., eavesdroppers) can monitor and record the communication between the UAs and the proxies. Proxychain protocol provides no additional information that passive attackers could use to impersonate users. Dictionary attacks against the challenge and the response values are still possible, but they require more effort than in Digest authentication due to the additional hash operations used in Proxychain.

Finally, Proxychain makes SIP authentication cheap enough to authenticate more than one message per session. Authenticating more SIP messages per session provides protection against several known attacks that target current SIP deployments. From the security perspective, all the messages should be authenticated to avoid vulnerabilities. Proxychain represents a first step in this direction.

6.3 Availability

The availability of the database is critical in scenarios with a central authentication service. For example, if the database becomes unavailable, the proxies will be unable to authenticate UAs requests. As a result, no call sessions can be established until the database is back online. This risk can be mitigated through mechanisms such as high availability clusters or backup sites. However, these alternatives are typically expensive and complex to manage.

Proxychain offers a cheaper alternative for database outages. The idea is that the database can create a list of authentication credentials with long enough hash chains and no expiration time. These backup-credentials can be stored offline in each proxy location and be activated when the database is not available. Once each proxy loads the backup-credentials in memory, they will be able to

authenticate UA requests as long as the credentials are active (sequence counter > 0). A naïve approach would be to generate backup-credentials with uniformly long hash chains (i.e., length = 1,000) to reduce the risks of users finishing their credentials before the database is back online. However, this approach is inefficient because very long hash chains will cause unnecessary overheads in the database and the UAs and lower performance during their generation. A more efficient approach would be to estimate the necessary length of the hash chains based on the expected time that the database is going to be unavailable. For example, a provider needs to install new hardware, requiring the database to be offline. The provider can estimate how many authenticated requests occur in a period of six hours based on its call statistics. For example, the provider can determine the call rate of its most active users. Assuming that the most active users make 10 calls per hour during busy hours, backup-credentials with a hash chain length of at least 60 will be required (also assuming that only one request per call is authenticated). Using Table 1, we know that the time to compute one credential with hash chain length = 100 is approximately 335 μsecs. Therefore, if the provider has 5 million users, the database will require approximately 28 minutes of computation to generate backup-credentials that will be active during 6 hours. This simple calculation could be made more robust by identifying those users most likely to far exceed the uses of the temporary credentials (i.e., profiling via long-term logging) and selectively increase the length of their hash chains.

7 Related Work

Authentication is a required service in most SIP deployments. The VoIP standard (RFC 3261 [21]) recommends the use of robust security mechanisms such as TLS, IPsec and S/MIME to provide authentication and other security guarantees. However, these mechanisms are computationally expensive [5, 6, 15, 16] and complex to manage (i.e., client certificates are required). Digest authentication, also recommended by RFC 3261, is more efficient and simpler authentication mechanism with lower implementation requirements than the previous schemes. As a result, it is the preferred authentication mechanism for most SIP deployments. However, previous research shows that Digest authentication can still have a considerable impact on the performance of a SIP infrastructure [19, 22], specially when a remote authentication database is employed [8]. In addition, in scenarios where the remote database is shared by multiple proxies, the database could become a bottleneck due to the high load the it receives. In this case, the database could be susceptible to DoS attacks. For example, multiple malicious clients could generate enough load to saturate the database, as was demonstrated by Traynor et al. [24] in cellular networks.

The impact on performance caused by authentication is also one of the reasons why only a few messages are authenticated in a SIP call transaction. This fact and the lack of mutual authentication lead to several possible attacks against a SIP infrastructure [2, 29]. Moreover, Digest authentication is considered a weak authentication protocol by today's cryptographic standards given its lack of mutual authentication and susceptibility to a number of other attacks [3, 9, 26, 27]. Several alternative schemes have been proposed to overcome the weaknesses of Digest authentication. Most of these alternatives focus on providing stronger security guarantees [4, 25, 27, 28], while others also focus on better performance [7]. However, these alternatives do not provide an implementation and experimental performance analysis to determine their impact on the performance of a SIP server.

To avoid the saturation of the database and improve performance, we present a new authentication mechanism based on temporary authentication vectors stored in the proxies. A similar idea is used in the Authentication and Key Agreement (AKA) [1] security protocol for 3G cellular networks. However, instead of using multiple authentication vectors as AKA does, our scheme uses a modified hash chain construction [13] to provide mutual authentication. Hash chains have been used in security protocols in different domains where efficiency is critical such as sensor networks [14, 20] and RFID tags [23]. Our work is the first to take advantage of the security, performance and space efficient properties of hash chains to reduce the overhead of the authentication process in SIP.

8 Conclusions

VoIP has and will continue to change telephony. These systems not only drastically reduce the costs associated with building and providing such services, but also offer the potential for rich new sets of features. Unfortunately, the large-scale usage of VoIP also creates a number of new security concerns. In this paper, we develop Proxychain, a mechanism that provides strong authentication between VoIP providers and their customers. Unlike previously deployed mechanisms, Proxychain is highly scalable and offers throughput improvements of greater than an order of magnitude. This increased efficiency allows providers not only to support a much larger customer base on a relatively limited hardware footprint, but also increases the overall security of the network by allowing for multiple message types to be authenticated. In so doing, we have significantly increased the robustness of VoIP systems.

Acknowledgments

We wish to thank Kevin Butler and our shepherd Galen Hunt for their valuable comments. This work was supported in part by the US National Science Foundation (CNS-0916047). Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] 3GPP. ETSI Technical Specification 133 102 v7.1.0 - Security Architecture, 2006.

[2] ABDELNUR, H., AVANESOV, T., RUSINOWITCH, M., AND STATE, R. Abusing SIP Authentication. In *Proceedings of the International Conference on Information Assurance and Security* (2008).

[3] BLACK, J., COCHRAN, M., AND HIGHLAND, T. A Study of the MD5 Attacks: Insights and Improvements. In *Fast Software Encryption* (2006).

[4] CAO, F., AND JENNINGS, C. Providing Response Identity and Authentication in IP Telephony. In *Proceedings of the First International Conference on Availability, Reliability and Security (ARES)* (2006).

[5] CHA, E.-C., CHOI, H.-K., AND CHO, S.-J. Evaluation of Security Protocols for the Session Initiation Protocol. In *Proceedings of 16th International Conference on Computer Communications and Networks (ICCCN)* (2007).

[6] COARFA, C., DRUSCHEL, P., AND WALLACH, D. S. Performance analysis of TLS Web servers. *ACM Transactions on Computer Systems 24*, 1 (2006), 39–69.

[7] CUI, T., GAO, Q., AND HE, B. A lightweight authentication scheme for Session Initiation Protocol. In *International Conference on Communications, Circuits and Systems* (2008).

[8] DACOSTA, I., BALASUBRAMANIYAN, V., AHAMAD, M., AND TRAYNOR, P. Improving Authentication Performance of Distributed SIP Proxies. In *Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm)* (2009).

[9] EL SAWDA, S., AND URIEN, P. SIP Security Attacks and Solutions: A state-of-the-art review. In *2nd International Conference on Information and Communication Technologies* (2006).

[10] FISHER, K., AND GRUBER, R. PADS: Processing arbitrary data streams. In *Proceedings of Workshop on Management and Processing of Data Streams* (2003), DIMACS.

[11] FRANKS, J., HALLAM-BAKER, P., HOSTETLER, J., LAWRENCE, S., LEACH, P., LUOTONEN, A., AND STEWART, L. RFC 2617: HTTP Authentication: Basic and Digest Access Authentication. *IETF RFC Editor* (1999).

[12] KAUFMAN, C., PERLMAN, R., AND SPECINER, M. *Network Security: Private Communication in a Public World*, second ed. Prentice Hall, 2002.

[13] LAMPORT, L. Password authentication with insecure communication. *Communications of the ACM 24* (1981), 770–772.

[14] LIU, D., AND NING, P. Multilevel μTESLA: Broadcast authentication for distributed sensor networks. *ACM Trans. Embed. Comput. Syst. 3*, 4 (2004), 800–836.

[15] MEENAKSHI, S., AND RAGHAVAN, S. Impact of IPSec Overhead on Web Application Servers. In *International Conference on Advanced Computing and Communications(ADCOM)* (2006).

[16] MILTCHEV, S., IOANNIDIS, S., AND KEROMYTIS, A. D. A Study of the Relative Costs of Network Security Protocols. In *Proceedings of the FREENIX Track: USENIX Annual Technical Conference* (2002).

[17] MITCHELL, C. J., AND CHEN, L. Comments on the S/KEY user authentication scheme. *SIGOPS Oper. Syst. Rev. 30* (1996), 12–16.

[18] MUKERJEE, S. Subscriber management for next-generation networks. `http://www.xchangemag.com/webexclusives/62h2815505.html`, 2006.

[19] NAHUM, E. M., TRACEY, J., AND WRIGHT, C. P. Evaluating SIP server performance. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (2007).

[20] PERRIG, A., SZEWCZYK, R., WEN, V., CULLER, D., AND TYGAR, J. D. SPINS: Security Protocols for Sensor Networks. In *Proceedings of the International Conference on Mobile Computing and Networks (MOBICOM)* (2001).

[21] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., AND SCHOOLER, E. RFC 3261: SIP: Session Initiation Protocol. *IETF RFC Editor* (2002).

[22] SALSANO, S., VELTRI, L., AND PAPALILO, D. SIP security issues: the SIP authentication procedure and its processing load. *IEEE Network 16* (2002), 38–44.

[23] SYAMSUDDIN, I., DILLON, T., CHANG, E., AND HAN, S. A Survey of RFID Authentication Protocols Based on Hash-Chain Method. In *ICCIT: Proceedings of the Third International Conference on Convergence and Hybrid Information Technology* (2008).

[24] TRAYNOR, P., LIN, M., ONGTANG, M., RAO, V., JAEGER, T., LA PORTA, T., AND MCDANIEL, P. On cellular botnets: Measuring the impact of malicious devices on a cellular network core. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (2009).

[25] WANG, F., AND ZHANG, Y. A new provably secure authentication and key agreement mechanism for SIP using certificateless public-key cryptography. *Computer Communications 31* (2008), 7.

[26] WANG, X., AND YU, H. How to Break MD5 and Other Hash Functions. In *Proceedings of EUROCRYPT* (2005).

[27] YANG, C.-C., WANG, R.-C., AND LIU, W.-T. Secure authentication scheme for session initiation protocol. *Computers and Security 24* (2005), 381–386.

[28] YOON, E.-J., AND YOO, K.-Y. A New Authentication Scheme for Session Initiation Protocol. *Complex, Intelligent and Software Intensive Systems, International Conference* (2009).

[29] ZHANG, R., WANG, X., YANG, X., AND JIANG, X. Billing attacks on SIP-based VoIP systems. In *Proceedings of the first USENIX workshop on Offensive Technologies* (2007).

Notes

[1] Note that calls are placed through "Super-Nodes" in Skype, but that users sign on through a server located at 80.160.91.11.

[2] `http://www.netlab.cc.gatech.edu/`

[3] `https://www.planet-lab.org/`

[4] `http://www.opensips.org/`

[5] `http://www.mysql.com/`

[6] `http://sipp.sourceforge.net/`

[7] Based on empirical evaluation, not shown for space reasons.

[8] R^2 is the correlation coefficient, which indicates goodness of fit, with 0 being no match and 1 being perfect.

[9] Setting a new hash chain once the current one expires, using a secure secondary channel