F3ildCrypt: End-to-End Protection of Sensitive Information in Web Services

Matthew Burnside and Angelos D. Keromytis

Department of Computer Science
Columbia University in the City of New York
{mb, angelos}@cs.columbia.edu

Abstract. The frequency and severity of a number of recent intrusions involving data theft and leakages has shown that online users' trust, voluntary or not, in the ability of third parties to protect their sensitive data is often unfounded. Data may be exposed anywhere along a corporation's web pipeline, from the outward-facing web servers to the back-end databases. The problem is exacerbated in service-oriented architectures (SOAs) where data may also be exposed as they transit between SOAs. For example, credit card numbers may be leaked during transmission to or handling by transaction-clearing intermediaries.
We present F3ildCrypt, a system that provides end-to-end protection of data across a web pipeline and between SOAs. Sensitive data are protected from their origin (the user's browser) to their legitimate final destination. To that end, F3ildCrypt exploits browser scripting to enable application- and merchant-aware handling of sensitive data. Such techniques have traditionally been considered a security risk; to our knowledge, this is one of the first uses of web scripting that *enhances* overall security.Our approach scales well in the number of public key operations required for web clients and does not reveal proprietary details of the logical enterprise network. We evaluate F3ildCrypt and show an additional cost of 40 to 150 ms when making sensitive transactions from the web browser, and a processing rate of 100 to 140 protected fields/second on the server. We believe such costs to be a reasonable tradeoff for increased sensitive-data confidentiality.

1 Introduction

Recent intrusions resulting in data leakages [20, 3] have shown that online users simply cannot trust merchants to protect sensitive data. Security incidents and theft of private data are frequent, often in spite of the best intentions of corporate policy, faithful compliance to standards and best practices, and the quality of security/IT personnel involved. Data may be exposed anywhere along a web-driven pipeline, from the outward-facing web servers to the back-end databases, so security personnel must protect a wide front. Furthermore, in service-oriented architectures (SOAs), data may also be exposed as they transit between SOAs, and, of course, the remote SOAs must also be configured and administered safely.

Data leakage can be very expensive to the parties involved. It was recently reported that an attacker compromised the networks of clothing retailer TJ

Maxx and stole credit card information for 45.6 million customers, dating back to December 2002 [20]. It is estimated that this breach will cost TJ Maxx approximately $197 million. Another attacker stole 4.2 million credit card numbers from grocery store chain Hannaford [3] with an unknown cost to the company, though a recent study [17] estimated an average cost of $197 per compromised customer record.

For a corporation to safeguard sensitive user information, it must be protected in an end-to-end fashion [25], in transit from the web browser to the back-end databases, and during storage at the database. Protecting the back-end databases may come in the form of legal [1] or technical [11, 8] protection. F3ildCrypt focuses on the technical protection of data in transit. While a protocol like SSL provides adequate protection for data on the wire, it provides no protection for transitions. Even with SSL protection between a web browser and web server, and between the web server and back-end database, sensitive data may still be revealed during the transition across the web server.

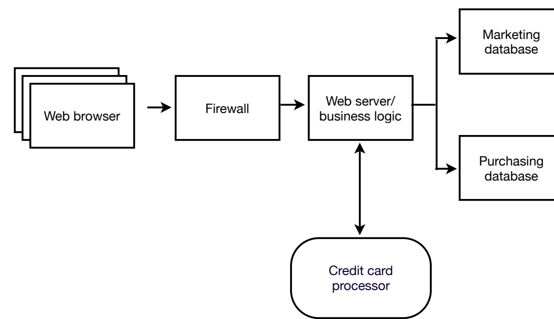

Fig. 1. A simple e-commerce server pipeline.

Consider a simple e-commerce website for a widget store, as in Figure 1. The website uses an Asynchronous Javascript and XML (Ajax)-based shopping cart [16] and XML-formatted content, served from PHP-based business logic. All customer communication with the website takes place over SSL. Customer data, including name, address, and order history are processed by the business logic and stored in a back-end MySQL database. As new orders for widgets arrive, the business logic transmits order information to the website's credit card processor.

An order consists of an XML document[1] containing the customer's name, address, email address, a list of each type of widget to be purchased, and the customer's credit card details. Note that each field is useful to only a subset of

[1] The choice of XML is not integral to our scheme; we can just as easily use JSON or any other data encoding/representation. XML was selected for convenience in prototyping, and because of its wide use in SOA environments.

applications in the website pipeline. That is, multiple machines have access to data for which they have no need – a violation of the principle of least privilege [26]. For example, there is no reason to expose the credit card data to the web server – in fact, it should only be revealed to the credit card processor – and there is no reason to reveal the customer's email address to the purchasing database.

To use this website safely, a customer must trust that both the widget store and the credit card processor are taking appropriate steps to protect his credit card information. Additionally, as far as the user is concerned, any protection derived from the SSL session is lost in the pipeline downstream from the web server, since the SSL tunnel ends at the web host. There is no guarantee to the customer or to the corporation that the downstream machines are not currently compromised and that they are suitably protected against future compromise (since downstream machines may be located in SOAs operated by third-party corporations).

The goal of F3ildCrypt is to guarantee that data are encrypted end-to-end, as they traverse an SOA and its children SOAs. F3ildCrypt is based around three components: an XML gateway, an in-line proxy re-encryption engine [7], and a Javascript policy and Java applet cryptographic engine.

We use an Ajax-based approach where fields are encrypted at the customer's web browser. In the straightforward approach, this would require that the Ajax application be bundled with certificates containing public keys for the internal web-pipeline components, so it can encrypt the information appropriately. However, this approach may not be appealing to a corporate entity, since it requires, for example, revealing the name and public key of the corporation's credit card processor. In general, it exposes the internal logic of the enterprise (including external business relationships, processing intermediaries, and the internal pipelines, which may change at any time) to the customer. A key contribution of this work is to use proxy re-encryption at the gateway to map fields encrypted by the user to the individual internal components or partner SOAs, without exposing clear-text at the gateway, and without revealing those partner relationships to the end-user.

Ajax-like techniques (and, more generally, web browser scripting) have long been considered a security risk, for good reasons. To our knowledge, our approach is one of few that enhances overall security through use of such techniques. Although the use of public key cryptography inevitably increases the overall latency and processing cost of any given web transaction, we experimentally demonstrate that the costs in this case are reasonable. Furthermore, these costs need only be incurred when sensitive information is being transmitted; in our widget-store testbed, the costs are only incurred when the user makes a purchase. The preceding portion of the session, while the user is browsing in the store, does not incur *any* additional performance impact.

2 Related work

Proxy re-encryption [9, 7] allows a third-party to transform a ciphertext for Alice into a ciphertext for Bob, without revealing the plaintext to the third party.

Consider Alice, with key pair (pk_A, sk_A), and Bob with key pair (pk_B, sk_B). A re-encryption key from Alice to Bob $rk_{A \to B}$ has the following property for all plaintext P:

$$pk_B(P) = rk_{A \to B}(pk_A(P)) \tag{1}$$

If, for example, Alice wishes to temporarily re-direct her encrypted email to Bob, but she does not wish to reveal her secret key, she can generate a re-encryption key $rk_{A \to B}$ and deliver that key to a proxy. (See [7] for the details on generating this key; it is a function of Alice's private key and Bob's public key.) The proxy can then re-encrypt messages destined for Alice so that Bob may read them. The plaintext is never revealed to the proxy.

The authors in [7] demonstrate a unidirectional, *single-hop* scheme, while the scheme proposed in [9] is bidirectional and *multi-hop*. Meaning, essentially, that $rk_{A \to B} = rk_{B \to A}$, and a ciphertext can be re-encrypted from Alice to Bob to Carol. The algorithm from [7] is partially implemented in the JHU-MIT Proxy Re-cryptography Library (PRL) [5], which we use in our prototype.

XML is fast becoming a standard for document transfer on the web, and there is a body of work on securing those documents. Element-level encryption of XML fields was first proposed by Maruyama and Imamura [24]. There are now several XML-based firewalls on the market including the Cisco ACE XML Gateway [4] and the IBM XS40 Security Gateway [6]. These devices allow field-level transforms, including cryptographic primitives, of XML content as it traverses the firewall. Appliances like these provide high performance, but do not provide end-to-end protection of the individual fields.

There have been a number of proposals for XML-based access control systems [14, 22, 15]. One of the most popular is the eXtensible Access Control Markup Language (XACML) [2]. It provides XML-based standards for defining policies, requests, and corresponding responses. An XACML policy consists of a list of subjects, actions and resources, followed by a list of rules for which subjects may apply which actions to which resources.

W3bCrypt [28] first introduced the notion of end-to-end encryption of data in a web pipeline. The W3bCrypt system consists of a Mozilla Firefox extension that enables application-level cryptographic protection of web content. Web content is encrypted or signed in the browser before being delivered to the web application. This provides field-level end-to-end protection for user data, but does not protect the corporate network from information revealed by the key distribution. That is, in order to use W3bCrypt across an entire web pipeline, including multiple possible calls to external SOAs, the logical architecture of the server network must be revealed to the client in the form of pairwise key sharing. By providing this protection, F3ildCrypt may be viewed as a successor to W3bCrypt.

Li *et al.* use *automaton segmentation* [21] to explore privacy notions in distributed information brokering systems. The authors model global access control

policies as a non-deterministic finite automata, and divide those automata into segments for evaluating network components. The automaton segmentation system considers privacy for users, data, and meta-data, but does not consider privacy notions with respect to the logical network layout and corporate interactions between service providers.

Sun Microsystems has implemented the Java WSDP 1.5 XWS-Security Framework [23] to assist programmers in securing web services. However, this scheme does not extend to the client (browser). Singaravelu and Pu [27] demonstrate a secure web services system based on the WS-Security framework. The key distribution mechanism used by this system requires pairwise shared keys between endpoints, potentially revealing the internal logical architecture and SOA dependencies. Chafle *et al.* [12] use data flow constraints to protect web services, but this requires complete, centralized control of all SOAs involved.

3 Architecture

In this section, we describe our network and threat models, and our design requirements. We then examine several design alternatives, before explaining the overall F3ildCrypt architecture.

3.1 Network model

We consider service-oriented architecture (SOA)-style networks where external requests to the network have a single entry point and request-handling takes the form of a tree. A single parent SOA may make requests on multiple child SOAs in the course of processing a request.The SOAs may each operate under different administrative domains, with varying legal and corporate policies toward the privacy and protection of data traversing their networks. There may also be political, corporate, or technical pressure to prevent disclosure of the logical architecture of each SOA, and the identities of their children SOAs.

3.2 Threat model

A corporation whose business model requires handling sensitive user information (*e.g.*, credit cards, Social Security numbers, *etc.*) has both financial and political incentives to protect those data as they traverse its network. There are commonly used mechanisms, like SSL, for protecting the data point-to-point, but this does not protect against data leakage at compromised intermediate hosts.

Thus, our threat model encompasses large-scale networks of inter-operating SOAs where multiple internal hosts or networks may be compromised. These nodes may cooperate to extract and reveal data from transient information flows. We focus particularly on those information flows containing sensitive data related to, *e.g.*, identity theft. Our approach does not protect against the compromise of a node that *legitimately* has the need to view a specific piece of sensitive information.

Additionally, the logical architecture of the corporate network, along with any SOA peering agreements, is sensitive. Information of this nature should be protected from disclosure.

3.3 Requirements

Our goal is to provide XML-field granularity end-to-end protection of data transmitted from a web browser to each field's destination end-host within the web pipeline of an e-commerce site. The web pipeline may encompass multiple remote SOAs, and the end-to-end property must hold across SOA boundaries. Additionally, the confidentiality of the logical internal architecture of each SOA must be respected. That is, no architecture details should be disclosed to the web clients or across parent or children SOA boundaries.

3.4 Design alternatives

An XML firewall, like those marketed by IBM [6] or Cisco [4], or a similar proxy, sited at an SOA's network edge, can provide some protection. The proxy or firewall encrypts individual fields of each document to the fields' destination host within the SOA. However, this is not an end-to-end solution and an end-user has no way of verifying that an XML firewall or proxy is in place, let alone operating correctly. The customer must simply trust the SOA beyond the narrow confines of the commercial transaction.

Another approach is to generate a public key pair at each host in the web pipeline, use a trusted third party (VeriSign, GeoTrust, *etc.*) to sign certificates for each, and deliver the certificate set to each web browser or SOA client. In the event that a document containing fields with sensitive data must be delivered to the website, the web browser (or a browser-embedded crypto engine) can then encrypt each field directly to its destination end host.

There are several serious flaws in this design. If the e-commerce site links to external SOAs, the keys for each host in each external SOA must also be delivered to the web client. Thus, this solution does not necessarily scale well in the number of certificates. As more SOAs become involved, a cache of hundreds or thousands of certificates would have to be provided to each new web client, and the certificate caches for existing web clients would have to be updated each time the internal architecture of the SOA or any of its dependent SOAs changed. This solution also has the unfortunate consequence of revealing details to the end user (and thus to competitors) about the logical architecture of the e-commerce site and its SOA partners. By collecting and correlating the certificate sets, an adversarial client may be able to identify individual hosts in an SOA. Furthermore, this technique reveals the identities of the SOA partners. These details may encompass trade secrets and other confidential information.

3.5 F3ildCrypt Architecture

Our proposed solution is based on the technique of proxy re-encryption. Each SOA publicizes a certificate containing a public key, called the external key, pk_E.

This key is used by the SOA's clients, either web browsers or other SOAs. Before sending a document containing sensitive data fields to an SOA, a client cryptographically transforms each field containing sensitive data, using the external key. The client chooses which fields to transform based on an XACML client policy delivered from the SOA.

Meanwhile, each host or application in the SOA has an associated public key pair. This set of public keys is the internal key set $pk_{I_0}...pk_{I_n}$. These keys are used for communication internal to the SOA.

The external key pk_E is generated at a host called the external-key holder. The public keys of the internal applications $pk_{I_0}...pk_{I_n}$ are delivered to this host and used, in concert with the external secret key sk_E to generate the re-encryption keys $rk_{E \to I_0}...rk_{E \to I_n}$, as in [7]. The fundamental property of proxy re-encryption holds that, for any plaintext P and internal application j:

$$pk_{I_j}(P) = rk_{E \to I_j}(pk_E(P)) \quad (2)$$

The re-encryption keys are installed at a host called the *proxy re-encryption engine*. Fields from documents arriving at the SOA have been encrypted under pk_E and are handled by the proxy re-encryption engine. The latter re-encrypts each field under re-encryption key $rk_{E \to I_j}$, where j is the individual host within the web pipeline designated to process that field, based on a XACML server policy. The plaintext *is not revealed* until it arrives at the intended destination host.

This solution requires an SOA to deliver to its clients a certificate containing only the single external key pk_E, avoiding the problem of sending what could be a set of hundreds or thousands of certificates. Furthermore, no logical infrastructure details are revealed to the client. With the exception of the external-key holder, any subset of intermediate hosts between the client and end-host – including the proxy re-encryption engine itself – can be compromised without leaking any sensitive user data.

Compromise of the external-key holder, however, could be dangerous, requiring that special care be taken to secure this machine. Luckily, the bandwidth requirements on the external-key holder are extremely low. It is only used to generate the re-encryption keys so, after initial setup, its use is only required when adding new internal hosts. Thus, in the extreme, it is possible to keep the external-key holder offline at all times, and distribute keys through it by hand.

4 Implementation

Our implementation of F3ildCrypt consists of a Javascript-based policy engine and a Java-based cryptography engine delivered to each web browser. The web server connects to the server using SSL. On the server side, we provide a Python-based XML gateway with in-line proxy re-encryption engine for each SOA, and a Python-based XML proxy at each internal application. These proxies store the key pairs for their respective applications, and perform decryption and XML unwrapping on behalf of the application.

The Java cryptography engine and in-line proxy re-encryption engine use the proxy re-encryption algorithm described in [7]. This algorithm is based on bilinear maps [10], and is partially implemented in the JHU-MIT Proxy Re-cryptography Library (PRL) [5]. For our implementation, we ported the PRL to both Java and Python. We note that the JHU-MIT PRL supports only single-hop re-encryption, thus limiting the recursive depth of our implementation until such time as an implementation of the multi-hop algorithm from [9] is available.

F3ildCrypt setup in an SOA begins by designating an offline host as the external-key holder and generating the external key pair. The public key pk_E is signed by a trusted third party and the certificate is made available to the public. This is the key with which all clients will encrypt sensitive data sent to the SOA.

At each application inside the SOA we install an XML proxy which serves as that application's entry point into the F3ildCrypt network. This proxy stores the internal key pair (pk_{I_j}, sk_{I_j}) associated with the application. Any documents with encrypted fields arriving at the application are intercepted and decrypted by the XML proxy before delivery to the application proper.

Each internal public key is delivered in offline fashion (hand-delivered via USB key, for example) to the external-key holder, where the re-encryption keys are generated. The re-encryption key for proxy j is $rk_{E \to j}$ and it is a function of the external secret key sk_E and pk_{I_j}. The re-encryption keys are then hand-delivered to the proxy re-encryption engine.

The proxy re-encryption engine operates as a client to the XML gateway. The XML gateway stores a set of XSLT stylesheets. Each stylesheet describes the transformation to be applied to a given field type in a document. The XSLT implementation is extended with the proxy re-encryption function, so applying the cryptographic transformations becomes an application of a stylesheet, as in W3bCrypt [28]. The specific stylesheets are chosen based on a system administrator-defined XACML policy.

The XML gateway uses the XSLT transforms to re-encrypt designated fields, targeting them to the appropriate internal hosts. It processes incoming documents containing fields encrypted under pk_E. These fields are re-encrypted under the various re-encryption keys $rk_{E \to I_0} ... rk_{E \to I_n}$, in accordance with the XACML policy, before forwarding the document on to the web pipeline.

When a client connects to the SOA over SSL, the SOA responds with the contents of an Ajax web application, implementing, for example, a shopping cart application. Packaged along with the application is the Javascript-based policy engine and an applet containing the Java cryptography engine. At the browser, the package then downloads from the SOA an XACML policy document to be applied to uploaded documents, and a certificate store containing the signed certificate for the SOA's external key. When, in the course of user interaction with the application, an XML document must be uploaded, the Javascript engine applies the XACML client policy. This policy describes which fields of the document should be encrypted. The cryptography engine encrypts the designated fields with the external key, and then the document is uploaded to the SOA.

Now consider the case of a parent SOA, with external key pk_{E_p} making requests on a child SOA with external key pk_{E_c}. The child SOA implements the F3ildCrypt architecture, with internal key pairs for its own internal applications. As in the parent case, and given the appropriate proxy re-encryption algorithm, XML documents arriving at the child SOA's XML gateway are re-encrypted by the proxy re-encryption engine.

To make use of the child SOA, the system administrator at the parent uses the publicly known pk_{E_c} and its secret key sk_{E_p} to generate a re-encryption key $rk_{p \to c}$. Fields within a document sent to the parent SOA, but destined for the child SOA, are re-encrypted under $rk_{p \to c}$ at the parent XML gateway. When the fields arrive at the child XML gateway, they may be re-encrypted again, to the end-hosts within the child SOA.

4.1 Example

In this section we will describe a sample application of the F3ildCrypt architecture. It is based on the network for a small e-commerce site selling widgets, called Widgets4Cheap. The site consists of a firewall, web server with business logic, and back-end databases for marketing and purchases, as was shown in Figure 1. Widgets4Cheap also makes use of an external credit card processor.

The website presents to the user a web page with a simple catalog and shopping cart application, where the user may browse widgets and select items to purchase. When the customer makes an order, the order is delivered to the web server in the form of an XML document. An order consists of the customer's name, physical address, email address, a list and count of each model of widget to be purchased, and the customer's credit card information.

Customer data, including name, billing address, and order history are stored in the purchasing database. The customer's email address is stored in the marketing database. As orders arrive, the business logic transmits order information and credit card details to the website's credit card processor.

Revealing the internal architecture of the Widgets network is undesirable, as it may reveal business or trade secrets (this is exacerbated in more sophisticated networks). Additionally, even with an SSL connection between the client and the web server, the compromise of any internal host in the Widgets4Cheap pipeline could be catastrophic to the company and its customers, since every internal host, particularly the firewall and web server, has access to all the customer information in transit.

To protect this network, we define a high-level security policy. The customer's billing address, and order details may only be revealed to the purchasing database, while the email address may only be revealed to the marketing database. The credit card information and total payment is revealed only to the credit card processor.

Before implementing this policy, we deploy the F3ildCrypt infrastructure, as shown in Figure 2. Co-located with each internal application is an XML proxy which stores the key pair for that application. This XML proxy serves to decrypt the incoming XML documents, and unwrap the XML as necessary. On a separate

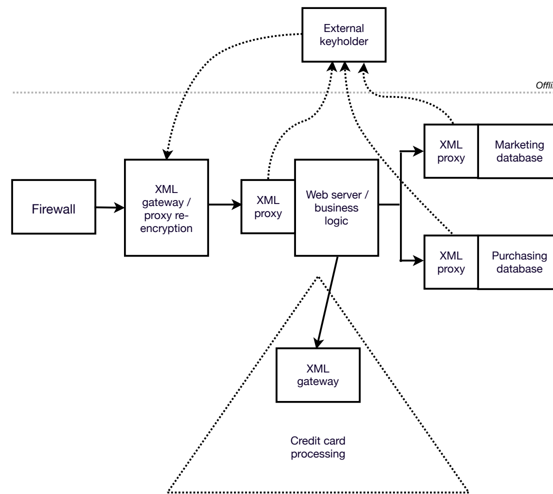

Fig. 2. Diagram of the network for Widgets4Cheap with F3ildCrypt installed.

offline machine (the external-key holder) the system administrator generates the external key pair which will be presented to remote users. A certificate for this key is signed by a third-party certificate authority. In the case of this example implementation, this was an in-house certificate authority.

The external-key holder is then used to generate re-encryption keys for each internal application and the credit-card processor, and these are delivered to the XML gateway, thereby allowing the gateway to re-encrypt traffic to the internal applications and credit-card processor SOA.

At the XML gateway we place a set of XACML policy files that describe the transformations to be applied to documents in transit, an example rule of which is shown in Figure 3. The XML gateway also contains a set of XSLT documents for implementing those transforms, an example of which is shown in Figure 4.

Meanwhile, the Javascript policy engine and Java crypto engine applet are incorporated into the Ajax application viewed by customers, along with a XACML client policy file and a certificate store containing the Widgets4Cheap external key. After browsing the catalog and selecting his items, the customer makes his purchase as in Figure 5. Before transmitting this document, the application applies the XACML client policy. The XACML client policy file describes which fields in the order document should be encrypted. A snippet from the Widgets4Cheap client policy is shown in Figure 6. When the policy is evaluated, the cryptography engine encrypts the necessary fields, resulting in a new, field-encrypted order document.

When the now-transformed document arrives at the Widgets4Cheap website,

```
<rule ruleid="creditcard_transform" effect="permit">
   ...
      <attributevalue datatype"string">
        order/creditcard
      </attributevalue>
   ...
</rule>
<obligation obligationid="reencrypt_on_transit"  fulfillon="permit">
  <attributeassignment attributeid="reencrypt" datatype="string">
    ccn_reencrypt.xsl
  </attributeassignment>
</obligation>
```

Fig. 3. A rule from the XACML server policy file. When the gateway receives an XML document, the rule attempts to match the XPath `order/creditcard`. When this rule fires, the obligation indicates that the XSLT transform `ccn_reencrypt.xsl` should be applied.

```
<xsl:template match="creditcard">
  <xsl:copy-of
      select="encrypt:reencrypt(., reencrypt_key[7]')"/>
</xsl:template>
```

Fig. 4. An XSLT snippet for re-encrypting the credit card information. Demonstrates usage of the XSLT extension function `reencrypt()`. It applies proxy re-encryption to the matched XML field using the re-encryption key `reencrypt_key[7]`.

it is processed by the XML gateway/proxy re-encryption engine, which applies the server XACML policy to determine which XSLT transforms to apply. The XSLT transforms apply the proxy re-encryption to the document, re-targeting the field encryptions that were originally applied by the client. The business logic then processes the order, delivering the various XML fields to their intended targets. The individual XML fields are intercepted by the XML proxies at each application and decrypted before being passed on to the application proper. The re-encrypted credit card information is passed to the credit-card processor, who may recursively apply this system, distributing the received information through its network.

5 Evaluation

We evaluated the performance of F3ildCrypt by measuring its impact on the web browser clients, on the XML gateway, and on the XML proxies at each host. We performed micro-benchmarks at the individual hosts, as well as throughput measurements on the servers.

Our experimental setup consisted of the network described in Figure 2. Each server application ran on a Dell PowerEdge 2650 Server, with a 2.0GHz Intel Xeon processor, 1GB RAM, and 36GB Ultra320 SCSI hard drive. All machines

```
<order>                                    <items>
<date>1 January 2008</date>                  <item>
<name>H. Simpson</name>                        <quantity>1</quantity>
<address>                                      <detail>Big red widget</detail>
  <street>742 Evergreen Ter.</street>          <cost>69.96</cost>
  <city>Springfield</city>                   </item>
  <state>USA</state>                         <item>
  <zip>12345</zip>                             <quantity>1</quantity>
</address>                                     <detail>Blue suede widget</detail>
<email>homer@springfield.com</email>           <cost>109.95</cost>
<creditcard>                                 </item>
  <payment>179.90</payment>                </items>
  <issuer>American Express</issuer>        </order>
  <number>1234-5678-1234-5678</number>
  <expiration month="10" year="2010"/>
</creditcard>
```

Fig. 5. A purchase order for two pairs of widgets from Widgets4Cheap.

```
<rule ruleid="creditcard_rule" effect="permit">
  ...
      <attributevalue datatype"string">
        order/creditcard
      </attributevalue>
  ...
</rule>

<obligation obligationid="encrypt_on_send" fulfillon="permit">
  <attributeassignment attributeid="encrypt" datatype="string">
    encrypt(key[n])
  </attributeassignment>
</obligation>
```

Fig. 6. The XACML client rule, abridged for clarity and space. This rule and obligation describes the action to be taken on the credit card section of the XML document: encrypting it with a key obtained from the certificate store.

ran OpenBSD 4.2. and were linked via Gigabit Ethernet. The applications included OpenBSD PF on the firewall, Apache 1.3.29/PHP 4.4.1 on the business logic server, and MySQL 5.0.45 on the database servers.

The client ran on a MacBook Pro, with a 2.4 GHz Intel Core 2 Duo, 2 GB RAM, and 150GB 5400 RPM Fujitsu hard drive. The machine used OS X 10.5.2 with Darwin kernel version 9.2.2. The web browsing platform installed on this computer was Mozilla Firefox 2.0.0.13.

The extra work incurred on the web browsing client consists of applying the XACML policy followed by application of the appropriate cryptographic transformations. We used a Java port of the JHU-MIT Proxy Re-cryptography Library [5], running as an applet in the browser, which implements the proxy re-encryption scheme described in [7]. The Java cryptographic engine applet and Javascript policy engine together are approximately 25KB. We measured the performance of the client by encrypting multiple 128-byte fields, as shown in

Figure 7a. After processing, most XML documents increase in size between 10% and 30%.

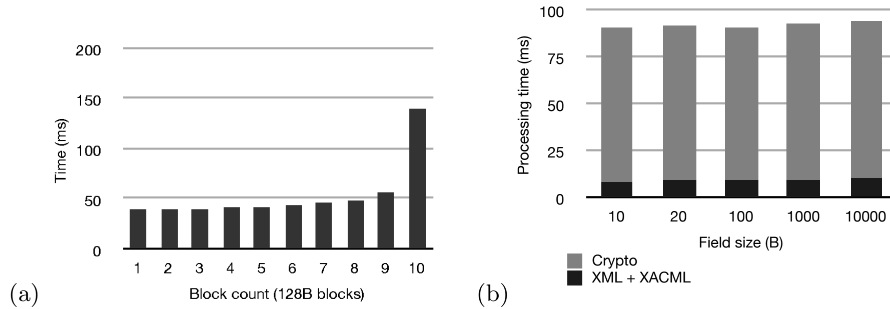

Fig. 7. (a) Time (ms) to encrypt multiple 128-byte fields on the client. **(b)** Processing time on a document containing a single field of 20 bytes. Shows the relative time devoted to cryptography versus the XML and XACML processing.

The most common sizes for identity-related sensitive data (*e.g.*, credit card numbers, birth dates, *etc.*) are less than 1K, so the cost incurred at the browser in these cases will range from 40 to 150 ms. Of course, this cost is only incurred when sensitive data requiring encryption is actually transmitted.

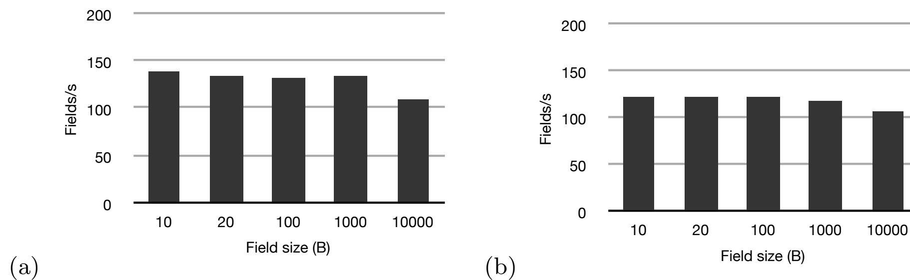

Fig. 8. (a) Re-encryption rate (fields/s) at the XML gateway vs. incoming field size. As field size grows, the processing rate decreases. **(b)** Decryption rate at an XML proxy vs. incoming field size.

The additional work incurred at the XML gateway consists of parsing the incoming XML documents and applying proxy re-encryption; Figure 7b shows the combined cost. We isolated the re-encryption cost per field in Figure 8a. An XML proxy decrypts the encrypted fields from incoming documents; we isolated the decryption cost at the XML proxy in Figure 8b.

These results show that fields from XML documents can be processed at

a rate of 100 to 140 fields/second, and the majority of the processing time is dedicated to the re-encryption process. This time can be significantly improved through software optimization; the JHU-MIT PRL is not optimized for execution time. The re-encryption cost can be further substantially reduced through the addition of a hardware cryptographic accelerator [19].

6 Discussion

The F3ildCrypt system is designed to assist an online entity in protecting its users' sensitive information. The user must not longer collectively trust the web application, the back-end databases, and the system administrators with each sensitive item he provides. Now, for that same item, he only has to trust its intended destination.

F3ildCrypt is designed to assist the system administrators in making the end-user's trust well-founded. However, to provide further assurance to the user, this approach may be combined with a P3P-like policy [13] working in concert with a browser-based cryptography engine like W3bCrypt [28]. Additional protection may come from obtaining a signature on the Ajax application itself from a trusted third party. This trusted third party (*e.g.*, the Better Business Bureau) would certify that the Ajax is encrypting or protecting data to the correct recipients. Regardless of the means, the user, or a trusted third party, must verify the contents of the Ajax application and the associated policy.

For a motivated adversary attacking a F3ildCrypt-enabled system, note that the external-key holder possesses the secret key corresponding to the external public key. Whoever possesses of the secret key is capable of decrypting all messages to that SOA, making the external-key holder a desirable target for attackers. However, it is infrequently used and has low bandwidth requirements. This machine can operate entirely offline, with the occasional generation of a re-encryption key taking place via diskette or USB key.

We also note that, within the network of the F3ildCrypt-equipped SOA, like in a traditional network, an adversary who has compromised an intermediate machine may swap or replay fields, or otherwise modify documents as they pass through that machine's possession. F3ildCrypt does not prevent such attacks, though they can be alleviated via timestamps and signatures on the individual fields.

There is an attack on web browsing transactions that comes from transaction generators. Transaction generators wait for users to log on to their accounts, and then issue transactions on their behalf. Jackson *et al.* [18] propose as a solution a form of confirmation page. This confirmation page can be integrated with F3ildCrypt and the user-certification process described above to provide additional protection to the user.

7 Conclusion

The F3ildCrypt system provides end-to-end protection to users and SOAs by encrypting XML fields at the client web browser. The SOA protects its internal architecture by using proxy re-encryption to re-target the XML fields at the SOA edge. The processing cost at the web browser ranges from .5 to 1 second when making sensitive transactions, and a processing rate of 100 to 140 XML fields/second on the server, of which the latter could be easily improved through software optimization and hardware acceleration.

Future work on F3ildCrypt includes integration of the proxy re-encryption engine with the web browser itself, and extensions to the browser to assure the user that the correct transformations have been applied.

Acknowledgements

This work was supported in part by the National Science Foundation through Grant CNS-07-14647 and by ONR through Grant No. N00014-07-1-0907. Any opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the NSF, ONR or the US Goverment.

References

1. Regulation (EC) No 45/2001 of the European Parliament and of the Council of 18 December 2000. Official Journal of the European Communities, December 2001.
2. OASIS eXtensible Access Control Markup Language (XACML). `http://www.oasis-open.org/committees/security/`, 2005.
3. Card data stolen from grocery chain. `http://www.securityfocus.com/brief/704`, March 2008.
4. Cisco ACE XML Gateway. `http://www.cisco.com/en/US/products/ps7314/index.html`, March 2008.
5. JHU-MIT Proxy Re-cryptography Library. `http://spar.isi.jhu.edu/~mgreen/prl/`, March 2008.
6. WebSphere DataPower XML Security Gateway XS40. `http://www-306.ibm.com/software/integration/datapower/xs40/`, March 2008.
7. G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *Proceedings of the 12th Annual Network and Distributed Systems Security Symposium (NDSS 2005)*, 2005.
8. Kun Bai, Hai Wang, and Peng Liu. Towards Database Firewall: Mining the Damage Spreading Patterns. In *Proceedings of ACSAC 2006*, pages 178–192, 2006.
9. Matt Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In *Proceedings of Eurocrypt '98*, pages 127–144, 1998.
10. Dan Boneh and Matt Franklin. Identity-based encryption from the Weil Pairing. *SIAM Journal of Computing*, 32(2):586–615, 2003.
11. Liang Cai and Xiaohu Yang. A reference model and system architecture for database firewall. In *Proceedings of IEEE SMC 2005*, pages 504–509, 2005.

12. Girish Chafle, Sunil Chandra, Vijay Mann, and Mangala Gowri Nanda. Orchestrating composite web services under data flow constraints. In *Proceedings of the IEEE International Conference on Web Services*, pages 211–218, 2005.
13. L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specifcation, April 2002.
14. Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(2):169–202, May 2002.
15. Irini Fundulaki and Maarten Marx. Specifying access control policies for XML documents with XPath. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 61–69, 2004.
16. Jesse James Garrett. Ajax: A New Approach to Web Applications. `http://www.adaptivepath.com/ideas/essays/archives/000385.php`, February 2005.
17. The Ponemon Institute. 2007 Annual Study: Cost of a Data Breach. `http://www.ponemon.org/press/PR_Ponemon_2007-COB_071126_F.pdf`, November 2007.
18. C. Jackson, D. Boneh, and J. Mitchell. Transaction generators: Root kits for the web. In *In proceedings of the 2nd USENIX Workshop on Hot Topics in Security*, 2007.
19. A. D. Keromytis, J. L. Wright, and T. de Raadt. The Design of the OpenBSD Cryptographic Framework. In *Proceedings of the USENIX Annual Technical Conference*, pages 181–196, June 2003.
20. Robert Lemos. TJX theft tops 45.6 million card numbers. `http://www.securityfocus.com/news/11455`, March 2008.
21. Fengjun Li, Bo Luo, Peng Liu, Dongwon Lee, and Chao-Hsien Chu. Automaton segmentation: A new approach to preserve privacy in XML information brokering. In *Proceedings of the 14th ACM conference on Computer and Communications Security (CCS)*, 2007.
22. Bo Luo, Dongwon Lee, Wang-Chien Lee, and Peng Liu. QFilter: fine-grained runtime XML access control via NFA-based query rewriting. In *The Thirteenth ACM International Conference on Information and Knowledge Management*, pages 543–552, 2004.
23. Qusay H. Mahmoud. Securing Web Services and the Java WSDP 1.5 XWS-Security Framework. `http://java.sun.com/developer/technicalArticles/WebServices/security/`, March 2005.
24. H. Maruyama and T. Imamura. Element-Wise XML Encryption. `http://lists.w3.org/Archives/Public/xml-encryption/2000Apr/att-0005/01-%xmlenc`, April 2000.
25. J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.
26. Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.
27. Lenin Singaravelu and Calton Pu. Fine-grain, end-to-end security for web service compositions. In *IEEE International Conference on Services Computing (SCC 2007)*, pages 212–219, 2007.
28. Angelos Stavrou, Michael Locasto, and Angelos Keromytis. W3bcrypt: Encryption as a stylesheet. In *Proceedings of the 4th Applied Cryptography and Network Security Conference (ACNS 2006)*, pages 349–364, 2006.

This article was processed using the LaTeX macro package with LLNCS style