Analysis of the Periodical Payment Framework using Restricted Proxy Certificates

Grigori Goldman **Lawrie Brown**

School of Information Technology and Electrical Engineering
University of New South Wales, Australian Defence Force Academy
Canberra 2600, Australian Capital Territory

grigori.goldman@adfa.edu.au
lawrie.brown@adfa.edu.au

Abstract

This paper discusses the design and implementation of a payment framework that is loosely based on the direct debit payment model. We define such payments as one in which customers can authorise merchants to bill them repeatedly for the provision of some service without further interaction with the customers being required. This paper aims to present a first working prototype of our periodical payment model, and to discuss its performance. Our model uses a novel approach for implementing security by employing X.509 restricted proxy certificates over Secure Socket Layer (SSL) to provide authentication, authorisation and non-repudiation services. Although the concept of electronic payments is hardly new and there is a significant amount of interest in improving its security model, most notably from Visa and MasterCard, periodical payments have been consistently overlooked by these industry heavyweights. As of now this concept remains unexplored and the current approaches for securing electronic transactions make it impossible to accommodate this transaction format. The work presented in this paper attempts to fill this niche by developing a new payment specification and a fully working prototype implementation addressing this issue.

Keywords: e-commerce, electronic payments, restricted proxy certificates, direct debit, mutual authentication.

1 Introduction

Despite considerable research effort by industry leaders like Visa and MasterCard into securing electronic transactions, there still appears to be little published work addressing significant niche topics like the support of periodical payments. In our opinion, most current solutions do not go far enough in addressing issues that are of importance to users of these payment applications. In the context of periodical payments, none of the new frameworks are capable of accommodating this transaction format. This is surprising considering that direct-debit remains one of the most popular payment methods available.

It is clear that current solutions being hastily implemented by industry stakeholders are aimed at fixing the most immediate problems in a quickest and likely cheapest way possible. There does not seem to be any effort spent on making these solutions “future-proof”.

Why is the current approach not sufficient? What limitations does it impose on its use? What alternative applications are needed? These are the questions that will be addressed in this paper in the context of the periodical payment framework.

Before we begin, however, it is important to establish precisely what we mean by this term *periodical payments*. Periodical payments represent direct debit transactions online. They are used for splitting a single logical transaction across multiple physical transfers based on an agreement between the merchant and its customer. This agreement is considered legally binding (although not currently automatically enforceable) on both sides.

Consider the following example. A customer registers with an Internet Service Provider (ISP) to gain access to its Internet broadband service. During the subscription process, the customer provides the ISP with a credit card number, which it can use to charge the customer on a monthly basis. The ISP informs the customer of the amount that will be debited from his account and when this will occur, for example, on the first business day of each month. Once this process is completed, the ISP can debit this account without any further customer intervention.

For registering customers online, this ISP has opted to provide a single payment method option to its customers (i.e. credit cards). While other account types require ISPs to collect signed direct debit request (DDR) forms offline, credit card information can be collected and used without such formalities.

In the present electronic environment this ISP is not alone. Most companies choose credit cards as a payment method to simplify online payments processing. Unfortunately for consumers, this empowers merchants with unprecedented abilities to conduct ad-hoc transactions mimicking direct-debit payments without the backing of formally documented customer consent.

The increased flexibility with relaxed security model offered by credit cards makes the electronic direct debit model vulnerable to abuse. For instance, using the above example, the ISP can easily charge the customer twice within the same month regardless of their initial agreement that stipulated only one payment transaction per month.

Periodical payments as discussed here are not meant to replace in any way the standard electronic payment mechanisms in place currently. However, the concepts and technologies adopted for its use can be easily generalised to cover all possible electronic transactions.

Furthermore, it might seem unreasonable to expect either Visa or MasterCard to address problems inherent in online direct-debit payments. Existing direct debit applications are not standardised and their use is varied between different merchants and the payment options they choose to provide. This makes solving this issue particularly difficult. However, the argument that we shall make here is that unless provisions for this extra functionality are added now it might not be possible (or at least not simple) to add them in the future. In the next section (Related Work) this topic will be described in more detail with specific focus on the repercussions of the current approaches.

The aim of this paper is to present a prototype of an electronic payment framework that implements a direct debit payment model. This prototype builds on our previous work presented three years ago in Goldman (2007). While that paper presented the theoretical model for periodical payments its weakness was the lack of simulation and test results analysis. This paper aims to fix the shortcomings of its predecessor by strengthening our argument through analysis of initial test data obtained via testing of this prototype. Full details of this work are also provided in Goldman (2009).

A lot has changed in the last three years since the publication of (Goldman 2007). There are more transactions performed over the Internet than ever before. The problems with current periodical payment solutions remain, despite growing interest in electronic payment security. This work hopes to fill this niche.[1]

To minimise the scope, this paper focuses primarily on credit cards, as they are the most widely used and recognised form of online payments. However, it should be noted that the strength of the new approach is in its ability to deliver consistent behaviour across all account types not just credit cards.

The rest of this paper is organised as follows. In Section 2 a brief overview of the current state of academic and industry research in this area is presented. A great deal of attention and focus is dedicated to the discussion of popular approaches taken by the industry in this area and their influence on periodical payments. Section 3 introduces the periodical payment framework specification and discusses its implementation by presenting the prototype. Special emphasis is placed on improvements to the original proposal discussed in (Goldman 2007). In Section 4 a detailed analysis of the performance of the prototype is presented. The focus of this analysis at this stage is on the impact of introducing SSL mutual authentication using restricted proxy certificates into the framework. Some discussion on using web services as a communication layer is also presented although it is not the main focus here. The paper concludes with a discussion of remaining work (Section 5) and a conclusion (Section 6). Section 7 contains acknowledgements and section 8 references.

[1] Legal issues surrounding the use of digital certificates and legislative requirements for electronic payments are complex and best left to the experts. As such, these aspects are considered out of scope of this paper.

2 Related Work

With at least three decades of continuous interest in electronic payments, this area has no shortage of related work to use as a starting point. However, given the limited scope of this paper and maintaining its theme, this section will focus primarily on recent proposals that are relevant to credit card payments.

Due to high-complexity of the issue and the extreme difficulty of deploying new concepts and technologies into the market, the pace with which innovation has been introduced into electronic payments has been particularly slow. The frameworks that have been successful at breaching this hurdle are more often than not based on the current technologies and practices (with all of their limitations) rather than being innovative. In fact, the most successful of these systems (e.g. PayPal) are nothing more than proxies into current financial networks.

The industry, however, does at times attempt to break away from the accepted status quo by introducing new and radical changes. Both Visa and MasterCard, for example, are currently in the process of implementing and deploying new and competing authentication technologies aimed at reducing credit card fraud. We will cover these technologies in detail later in this section.

Before introducing competing products solving credit card fraud issues, both Visa and MasterCard (along with several industry leaders) worked together in designing a fundamentally new electronic payment protocol for credit card transactions over the Internet. In the late 1990s a Secure Electronic Transaction (SET) specification was developed. With enormous financial support and initial strong interest in this technology SET was off to a good start.

It has been over a decade since Visa and MasterCard have unveiled SET. However, despite all their effort this technology failed to gain acceptance within the industry. SET was abandoned to the extent that even obtaining references to its original specification documents is becoming difficult. No references to this protocol exist on either Visa or MasterCard web sites and the www.setco.org site dedicated to it has been disabled.

There is no doubt that SET was and most likely still is the most ambitious and technically advanced specification dealing with electronic payments. Despite its failure there is a lot that we can learn from it. By studying its design and by analysing the reasons for its failure we can avoid making the same mistakes.

SET specification defines two related phases: 1). registration phase, and 2). purchase phase. Consisting of approximately one thousand pages this protocol is extremely complex and would require an entire paper dedicated to its analysis. As such, only a brief overview will be presented here (see MasterCard, Visa 1997a and 1997b for details).

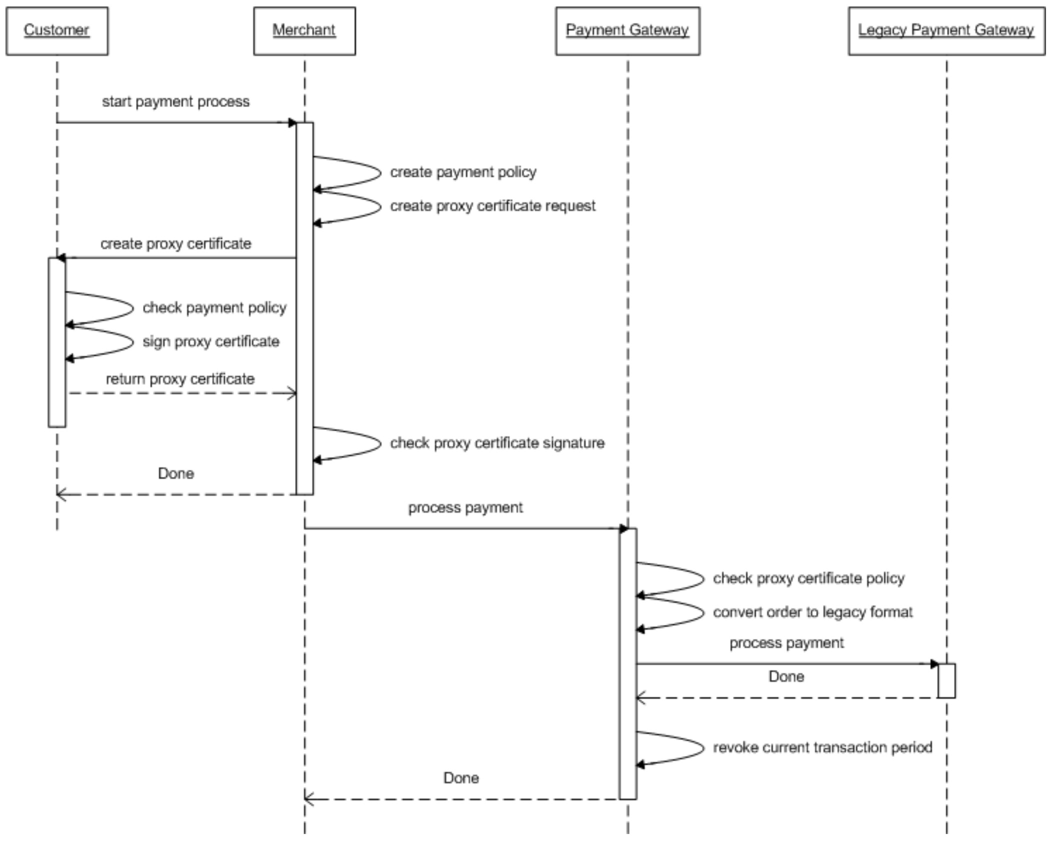

Figure 1: Proxy Certificate Delegation and Payment Process

The registration phase of the SET protocol deals with how both customers and merchants obtain cryptographic credentials that they can use for authentication and authorisation purposes. This is a mandatory step that both parties have to complete before they can participate in SET transactions. Registering for SET involves using a dedicated Certification Authority (CA) capable of issuing digital certificates on behalf of the card brands.

The purchase phase of the SET protocol is a more interesting although unnecessarily complicated process. Just like the framework presented in this paper, SET is concerned with payments only, rather than the entire shopping experience. As such, it assumes that by the time it is invoked the customer selections have already been propagated to the merchant.

The crux of this phase involves the use of client-side software. It initiates the payment process, creates order (OI) and payment instructions (PI) and performs various cryptographic and communication services. An important part of this process is protecting the OI from the payment gateway and the PI from the merchant using a dual signature concept introduced in (MasterCard, Visa 1997a). A dual signature is constructed by separately hashing OI and PI constructs and then concatenating the hashes, hashing them again and signing them with the customer’s private key.

When the merchant receives the request it contains the OI and the hash of PI as well as the customer’s signature. The PI itself is encrypted using a key only available to the payment gateway. The merchant can verify the signature without having access to the contents of PI. Conversely, by giving the payment gateway the encrypted PI and the hash of OI it can also verify the signature without gaining access to the OI.

The most interesting aspect of the SET protocol directly relating to periodical payments is its concept of instalment payments. Instalment payments allow a merchant to split a single large payment over multiple smaller transactions. It works by adding an optional construct into the PI, which contains the terms for each instalment similar to the proxy policies discussed later on. Since normally PIs cannot be used multiple times the payment gateway issues a separate authentication token (e.g. reuse token) to the merchant for the next scheduled transaction. Once the last payment is processed no more authentication tokens are issued.

There is no doubt that SET was ahead of its time. Despite all the effort dedicated to its development it unfortunately ended in failure. There are numerous reasons why it failed. For example, both the registration and the purchase phases of the protocol were extremely complicated requiring over a thousand pages of formal specification to properly describe them. This was frequently criticised in (Giampaolo et. al. 2002 and 2003, Paulson 2002, Stallings 2006).

In addition, SET required the use of downloadable client-side software, which contributed to frequent shopping cart abandonment. This, of course, is less relevant now due to the increases in bandwidth available to Internet users.

Also, its overuse of digital envelopes at every stage of the process, requiring generation of multiple symmetric keys, followed by continuous verification of signatures made SET unacceptably slow. For example, Wolrath (1998) described how some vendors reported lag times of up to fifty seconds for processing a single cardholder request. The solution presented later on avoids these performance issues by reducing the complexity of the protocol and minimising the use of cryptography.

As was already mentioned both Visa and MasterCard have abandoned SET. Instead they have introduced alternative, competing technologies aimed at solving credit card fraud issues. For Visa this technology is Visa Three Domain (3-D) Secure authentication protocol, while MasterCard implemented a Secure Payment Application (SPA) protocol. A good discussion of each protocol is provided by GPayments (2002).

Visa 3-D Secure uses browser redirection as the main mechanism by which cardholders are authenticated. Merchants redirect users to their issuers, which perform authentication and redirect the user back to the merchant site. This works particularly well using username and password but can also be adapted for use with smartcards.

Unfortunately, credential caching is impossible with browser redirection. As such, the cardholder must be authenticated for every transaction (meaning the customer needs to re-enter credentials over and over again). MasterCard SPA solves this issue using a client side applet.

Unlike SET, a SPA applet does not require any customer specific information hence it can be safely installed and used on even unsecured machines (e.g. Internet cafes, etc). Cardholders do need to enter issuer information before making purchases as it is needed during authentication.

Using an applet instead of browser redirects means that customer credentials can be cached locally and reused over multiple payment sessions. Once shopping is completed the credentials are destroyed and another user can safely reuse the SPA applet.

The design of Visa 3-D Secure and MasterCard SPA has a significant impact on the way periodical payments can be implemented. Despite their ability to use strong, two-factor authentication via digital certificates, both protocols are essentially session based. They provide no support for delegation and in fact require the customer to be physically present during every transaction.

Using the current approach taken by both vendors it is not possible to implement periodical payments since this particular transaction type precludes customer participation after the initial credential exchange.

3 Periodical Payment Framework

3.1 X.509 Proxy Certificate Definition

X.509 certificates are used to solve authentication and authorisation problems. A certificate binds a public key to a user identity, while the private key is kept secret. Using the private key, the user may prove to other parties that he is the owner of the certificate.

It is clear that reliance on the private key as proof of ownership of the certificate can be problematic if the user wishes to delegate some of his privileges to other parties. Without revealing the private key, this is not possible using the conventional X.509 certificate.

The proxy certificate extensions attempt to overcome this problem by providing mechanisms for delegating certificates to other parties without compromising the user's private key. These extensions are described in (Tuecke et. al., 2004).

Restricted proxy certificates allow issuers to define restrictions, which are placed on proxy certificates via policies. This enables issuers to define which subsets of their overall permissions are to be transferred to another party, thus subsequently reducing the potential damage that can be caused by credential misuse.

3.2 Periodical Payment Model Specification

Unlike SET that described a complete cardholder and merchant registration process, our periodical payment framework deals solely with payments. It assumes that cardholders and merchants will be issued with digital certificates separately (e.g. when being issued credit cards, etc). This makes our specification a lot simpler to describe and analyse.

In this section we shall present a brief overview of the theoretical model that underpins periodical payments. For a more comprehensive description refer to our previous work in Goldman (2007).

Periodical payment model consists of two separate processes. The first is the delegation of permissions by the customer to the merchant using restricted proxy certificates. The second is the actual payment transaction, initiated by the merchant that uses a proxy certificate to authenticate itself to the payment gateway on customer's behalf and perform the funds transfer.

The delegation of the restricted proxy certificate is the only step in this process that involves the customer. It is based on a well-known X.509 proxy certificate delegation protocol defined in (Tuecke et. al., 2004). It works by requiring the merchant to generate a proxy certificate request containing the policy, which is then forwarded to the customer. Once the customer signs this request, it becomes the proxy certificate used for initiating payment transactions.

The periodical payment policy used within the proxy certificate is the core of our payment model. It is used to encode the contractual agreement between the customer and the merchant and is subsequently used to verify legitimacy of each payment transaction. It is encoded as an XML document within the proxy certificate and generally declares one or more assertions that restrict the use of the certificate and the amount that a merchant can charge the customer.

Since proxy certificates are self-contained and are not co-signed by either the merchant or the payment gateway, we have used an attribute within the policy, *merchant-dn*, to bind the certificate to a specific merchant. This protects the certificate from being stolen and used by an unauthorised party.

The syntax of the policy language is simple. The <pay> element encapsulates most of the contractual information discussed in this paper. This assertion encodes both the contract boundaries (i.e. begin and end dates) as well as the interval between each transaction.

The example XML in Table 1 defines a simple policy that allows a merchant to charge a customer account a fixed amount of ten Australian dollars. The "on" attribute specifies when each transaction can occur. The syntax for this attribute is based on a common UNIX time-based scheduling service called CRON. In this particular example the CRON expression states that the merchant may initiate a transaction on the first workday of every month in 2009.

```
<payment-policy merchant-dn=”cn=ISP,…”
  payment-gateway-dn=”cn=PaymentGateway,…”>
   <source-account>
      <creditcard cardnumber=”xxx”
            cardholder=”name” expiry=”2011-10”/>
   </source-account>
   <pay currency="aud" amount="10"
          on="0 0 * 1W * ? 2009"/>
</payment-policy>
```

Table 1: Periodical Payment Policy XML

A customer will receive this policy as part of a proxy certificate request generated by the merchant. It is customer’s responsibility to check the terms of the agreement and if acceptable sign the request thus creating a proxy certificate which is then returned to the merchant. Once received, the merchant can verify the proxy certificate signature to ensure that it matches a valid cardholder. Provided that the signature is valid the proxy certificate can then be used to initiate payment transactions according to the payment policy. This process is depicted on the left hand side of Figure 1.

The process of initiating a payment transaction is simple. Using mutual authentication over Secure Socket Layer (SSL) protocol with X.509 restricted proxy certificate as the authentication credential, the merchant can unambiguously identify itself to the payment gateway and at the same time prove that it has rights to access a customer’s account. The advantage of using proxy certificates is that they are virtually identical to the standard X.509 certificates currently used for SSL. This means that this protocol can be used unmodified with restricted proxy certificates as authentication credentials.

Once authentication is completed and a secure channel is established the payment gateway can retrieve the payment policy from the certificate and use its assertions to determine the validity of the payment instruction it received from the merchant. The process of verification is also simple. The payment gateway must ensure that the amount being transferred is within the accepted limit. In addition, it will check that the time of the transaction matches the CRON expression inside the “on” attribute of the <pay> element. Provided that the transaction is valid, the payment gateway performs the transfer using the existing banking network and sends an acknowledgement message back to the merchant containing a transaction receipt. This process is likewise depicted in Figure 1. The key point to note is that merchant to payment gateway communication occurs without any customer intervention.

3.2.1 Double Charging Problem

There is a significant flaw in the above process. It assumes that given a policy, the payment gateway will be able to unambiguously determine whether a transaction is valid. In actual fact, it cannot. By only comparing a policy to the current payment instruction received from the merchant, it is not possible to detect double charging. That is, the payment gateway can only verify whether payment instructions match a policy but cannot say with any degree of certainty that the order is unique and has not been already processed previously.

This is one of the most fundamental issues that prompted our initial interest in this topic. The success of this framework is dependent on finding an appropriate solution to this problem.

The solution that we have adopted is based on the concept of certificate revocation as described in (Housley et al., 2002). Normally, certificate revocation lists are used to revoke compromised certificates. However, in our case proxy certificates cannot be revoked, as merchants will be unable to initiate any more payment transactions using them. Instead, the payment gateway can use the payment period defined within the policy to revoke the use of a certificate for the immediate payment period. For example, given a monthly payment period (as in Table 1), after processing a transaction the payment gateway would make an entry in the transaction revocation list (TRL) revoking the use of the presented certificate until the next month. Any attempt to use the same certificate twice within the same month would fail due to this entry in the list. The following is an example of such an entry in the TRL revoking the use of a certificate in the month of March. This entry is derived from the policy in Table 1.

```
revoke="* * * 1W MAR ? 2009"
```

3.2.2 Cancelling Periodical Payments

One important advantage of being able to electronically create and sign periodical payment policies is the ease with which such agreements can be cancelled when things go wrong. There are numerous reasons why policies may need to be cancelled. Due to long-term nature of these types of payments it is only natural to assume that customers may want to terminate agreements when their terms are no longer suitable (e.g. merchant competitors are offering better deals for the same type of services, etc).

Currently, cancelling direct debits once they are established is difficult. The control over this process rests in the hands of the merchants. Usually, it is not in their best interest to allow customers to terminate contracts before they expire due to the loss of potential income. As such, a lot of merchants make it difficult for customers to cancel agreements.

The periodical payment framework that we developed provides a solution to this problem by implementing a cancellation process based on an XML cancellation policy embedded within the overarching payment policy.

The merchant is responsible for encoding the conditions under which policies can be cancelled by making a special entry (i.e. <cancellation-policy>) within the payment policy XML. For example, Table 2 depicts an extension to our original payment policy listed in Table 1. In this example, two separate <pay> assertions are used to declare two distinct fee structures, which are dependent on when the policy is cancelled. Terminating the policy within the first six months in 2009 carries with it a higher cancellation fee of twenty dollars, while in the last six months this fee is halved.

Using this policy a customer may cancel the periodical payment agreement by communicating directly with the payment gateway using its URL (as encoded in the cancellation policy). The cancellation request must contain the original, signed proxy certificate and must be authenticated using the same certificate that was used to create the proxy. Using this information the payment gateway will be able to establish wether the cancellation request is legitimate. Provided that cancellation is permissible within the current payment period as per the policy, the payment gateway will process this request.

```
<payment-policy>
  …
  <cancellation-policy url=http://pg/cancelservice>
    <pay currency="aud" amount="20"
          on="* * * * 1-6 ? 2009"/>
    <pay currency="aud" amount="10"
          on="* * * * 7-12 ? 2009"/>
  </cancellation-policy>
</payment-policy>
```

Table 2: Periodical Payment Cancellation Policy XML

In our scheme the policies are not immediately cancelled when customer requests are processed. Since requests are most likely to be received in the middle of a payment period, merchants are entitled to a final transaction, such as receiving any outstanding amounts and charging fees. As such, the payment gateway simply registers each request (i.e. policy) in a cancellation registry to be processed next time that certificate is used by the merchant, at which time the merchant is given an opportunity to finalise the account.

This scheme makes a number of assumptions about how cancellation of periodical payments will be commonly used. These assumptions compromise between flexibility, functionality and simplicity of the protocol. For example, this scheme does not support immediate termination of contracts. This means that given a monthly payment period, a customer cannot suspend payments halfway through the month. In this case, this approach favours the merchant as it can collect more revenue before the contract is permanently cancelled. On the other hand, the merchant using this scheme is not aware that a policy has been cancelled until an attempt to debit a customer is made. Furthermore, the merchant must react immediately when it receives a pending cancellation message, hence all its administrative tasks must be easily automated. This may not appeal to some merchants, for example, merchants whose customer retention strategies yield good results.

Finally, to improve security this scheme binds the policy to a payment gateway (see *payment-gateway-dn* attribute in Table 1). This ensures that the merchant cannot switch payment gateways to bypass a registered cancellation. Unfortunately, this eliminates the possibility of load balancing or failover between different payment gateways. We recognise this as a limitation of the scheme and plan to address this in the future.

3.3 Payment Framework Implementation

The architecture of the periodical payment framework consists of three distinct tiers: 1). client-side browser extension, 2). merchant payment processing service, and 3). acquirer payment gateway web service.

Using a browser extension to implement the client-side software needed for enabling proxy certificate delegation is a logical choice. Browser plug-ins and extensions are becoming increasingly common way of distributing custom software to Internet users. This trend is partially motivated by the growing popularity and development of the Mozilla Firefox browser. This browser is built on top of a mature, extensible framework that makes it simple to dynamically extend its graphical user interface and behaviour.

Utilising a combination of XML User Interface Language (XUL), JavaScript and Java we have developed a simple prototype extension that captures HTTP requests carrying base64 encoded proxy certificate requests and returns signed proxy certificates. For now this extension looks for a user's keystore on the file system when signing proxy certificates, however, in practice Firefox provides features for accessing user credentials directly from smartcards.

The business logic is implemented using a combination of JavaScript and Java. JavaScript is used for processing HTTP requests and sending HTTP responses while all cryptographic operations are performed in Java. An optimised version of this extension would see Java being replaced with a more efficient XPCOM/C++ implementation.

The merchant software is implemented as a Java 2 Enterprise Edition (J2EE) web application running on Apache Tomcat servlet container. The communication between the merchant application and the client-side browser extension is handled by a servlet filter which encodes the logic for generating proxy certificate requests and processing proxy certificates returned by client browsers. This filter is designed to cache the new proxy certificates and their corresponding public/private key pairs for use by the merchant application (i.e. when it is ready to use them for initiating payment transactions).

Normally, a single Java web application would only use one digital certificate for SSL authentication. As such, a Java application can be configured at runtime with a keystore location using the following system property: javax.net.ssl.keyStore. This keystore is accessed during SSL mutual authentication process.

For periodical payments this default Java behaviour is insufficient, as a single Java process (i.e. merchant server) must dynamically change digital certificates when processing payments for different customers. To overcome this limitation a custom security provider and key manager was implemented that is capable of dynamically selecting digital certificates and their keys according to the currently executing customer payment. It can do this even in a multi-threaded environment. These credentials are obtained from a cache managed and updated by the servlet filter described previously.

Specifically, the merchant payment process is forced to use restricted proxy certificates when exchanging SOAP messages with the payment gateway web service. This web service was configured to be accessible only via HTTPS protocol (i.e. over SSL secured channel). Furthermore, mutual authentication was also enabled requiring the merchant to present a valid X.509 certificate during the SSL handshake.

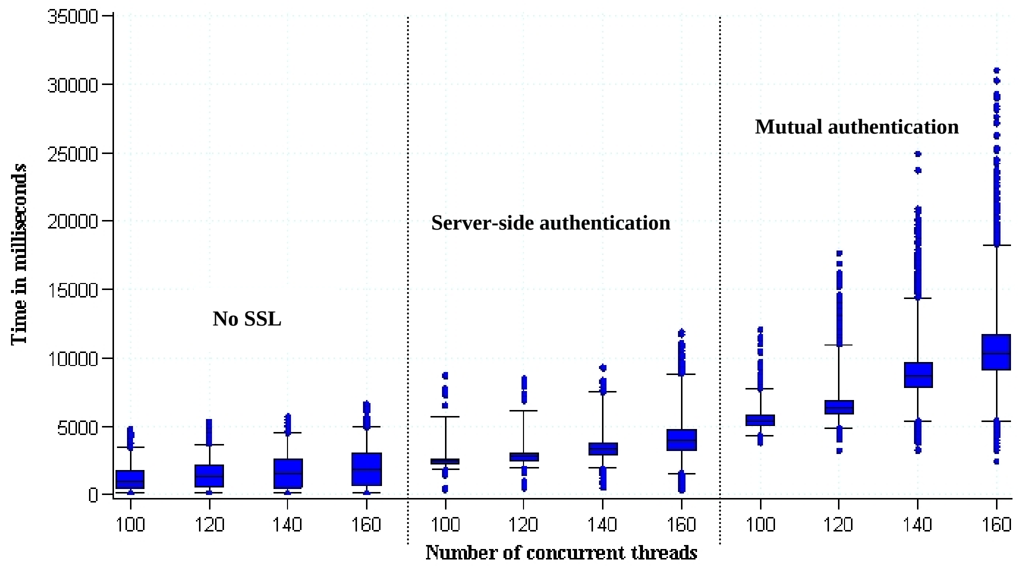

Figure 2: Total request processing time

Just like the merchant, the payment gateway application is deployed on Apache Tomcat servlet container. It uses Apache Axis2 as a web services engine that implements the entire web services feature stack.

During integration testing we discovered that the standard Java SSL socket implementation does not handle proxy certificates. It rejects proxy certificates because they contain an end-entity certificate within the certificate chain. In other words, it assumes that only CAs can issue and sign certificates. Due to time constraints we used a “work around” by adding a basic CA constraint to customer certificates thus in a fact making each customer a CA.

This is not a significant issue since there are alternative configurations that can make Tomcat recognise proxy certificates. Instead of using a Java Secure Socket Extension (JSSE) as an SSL provider, for example, we can use an Apache Portable Runtime (APR) that directly integrates with OpenSSL (SSL implementation that supports proxy certificates). Considering that OpenSSL is significantly faster than JSSE, changing SSL implementations should deliver better performance results making this change our top priority as we move forward.

4 Performance Analysis

A quantitative approach was chosen for testing the periodical payment prototype. Having little reference data to start with our aim was to collect enough empirical data to make reasonable conclusions regarding its overall performance. It is reasonable to assume that other electronic payment applications can benefit from the data collected here since many new applications are beginning to follow the same approach. For example, PayPal and Google Checkout are both introducing web services over SSL.

In this section we shall focus on the performance of the merchant to the payment gateway communication. We consider this part of the framework to be more critical than the relatively low volume interactions between the client browser extension and the merchant. This exchange is of course equally important; however, due to its infrequent use even relatively poor performance is likely to be acceptable.

Overall, four independent tests were executed with an increasing number of concurrent threads accessing the payment gateway web service. Specifically, the merchant service created 100, 120, 140 and 160 concurrent threads each one generating 100 independent payment requests thus producing in total 10000, 12000, 14000 and 16000 unique web service requests.

Each test was initialised with a set of pre-generated unique proxy certificates needed for SSL mutual authentication. Unique certificates were needed to ensure that the merchant or the payment gateway could not cache credentials and reuse them for multiple sessions. Each certificate contains a unique distinguished name and a policy whose amount value is also different to other policies within the test set.

Unfortunately, it was not possible to broaden the spectrum of tests by increasing the number of concurrent threads (or the number of requests per thread) due to the limitations imposed by the hardware. Any attempts to increase the load on the payment gateway caused crashes of the JVM. It remains an exercise for the future to re-run these tests using more capable hardware, that better reflects that likely to be used by merchants and payment gateways.

To be able to accurately measure the true impact of adding mutual authentication with restricted proxy certificates to the framework the same tests were executed with no SSL and with SSL server-side certificate authentication only.

The test configuration with no SSL authentication was useful for establishing a baseline/benchmark to compare other results with. On the other hand, server-side certificate authentication test is needed since this form of authentication is prevalent on the Internet today. Comparing the results of these tests against our proposed solution should make it easy to determine whether using certificate delegation for client-side authentication is a

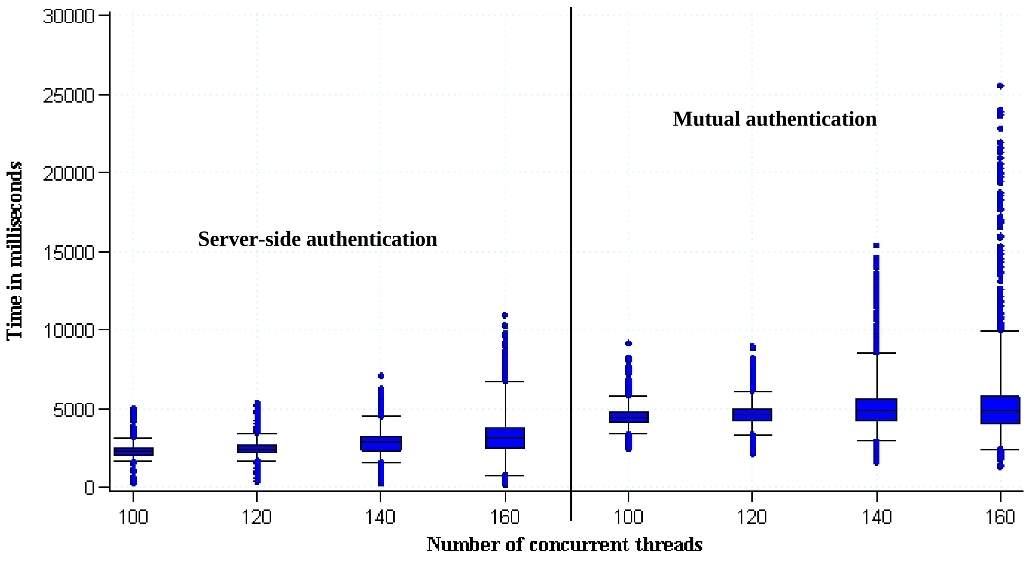

Figure 3: SSL handshake processing time

viable approach for solving the periodical payment problem.

4.1 Test Results

It is reasonable to assume that adding SSL to the framework will have a significant impact on its performance regardless of whether it uses server-side only or mutual authentication. Furthermore, we predict that increasing the load on the server by raising the number of concurrent threads will have a similarly detrimental effect. At best, we are expecting linear performance degradation as the number of simultaneous connections grows.

The results of executing the three test scenarios described in the previous section (i.e. no SSL, server-side and mutual authentication) are summarised in Figure 2 using a box plot. We have chosen to use a box plot, as it is a convenient and efficient way of summarising a large data set. It provides a visual representation of the most important aspects of a distribution (e.g. median value, inter-quartile range measuring statistical dispersion and lower/upper outliers).

Specifically, Figure 2 plots total request processing time of executing each request. An individual data point represents request processing time starting from when the merchant first connects to the payment gateway up to and including receipt of a response message upon completion of a transaction.

The results presented in Figure 2 are consistent with our prediction that the growth in request processing times is linear. This observation can be mathematically substantiated using Pearson’s Product Moment Correlation Coefficient (PMCC). A PMCC value of 0.9975 (for no SSL authentication), 0.9877 (for server certificate authentication), and 0.9895 (for mutual authentication) reveals an almost perfect linear correlation between the number of simultaneous connections and the performance of the server. The fact that this is happening consistently regardless of the changes to SSL configuration increases our confidence that our observations are accurate.

Unfortunately, increasing the number of simultaneous requests impacts data dispersion. The inter-quartile range (IQR) within each data series rises with the number of concurrent threads. Containing 50% of all requests, these IQR measurements indicate an increase in volatility as the server is overloaded. It should be noted, however, that this in itself is not unusual. As the load on the payment gateway increases, environmental factors are starting to play a more significant role on performance of individual requests thus creating variability in test results (e.g. garbage collection, page swapping, etc).

Figure 2 also reveals that there is a substantial increase in request processing time using mutual authentication. This difference is much higher than the original variation between no SSL and server-side certificate authentication only scenarios.

In Figure 3 the impact of mutual authentication on performance is measured in relation to the standard server-side certificate authentication. Unlike Figure 2, which depicts the total processing time for each individual request, this graph shows only the SSL handshake part of the overall request execution logic. By isolating the SSL handshake process we eliminate the ambiguity introduced by other parts of the application such as the web services engine that could have skewed the results.

Figure 3 follows the pattern established in Figure 2. The data represents a distribution with a flat curve and a positive skew. This data is also similarly dispersed with the number of concurrent threads having the same impact as before. That is, increasing the spread of data over a larger time set. However, the rate with which performance degrades does not match. Figure 2 displays a much higher rate of decay than Figure 3 (87 milliseconds per additional thread versus 10 milliseconds). This imbalance suggests that other factors unrelated to mutual authentication are affecting performance.

We can confirm that SSL is not solely responsible for degrading performance as Figure 2 suggests by examining the ratio of SSL handshake to total time it takes to process a single request. In Figure 4 this ratio is described as the percentage of the difference between

SSL handshake and the total time. This data is plotted using the average percentage over segments of 1000 requests.

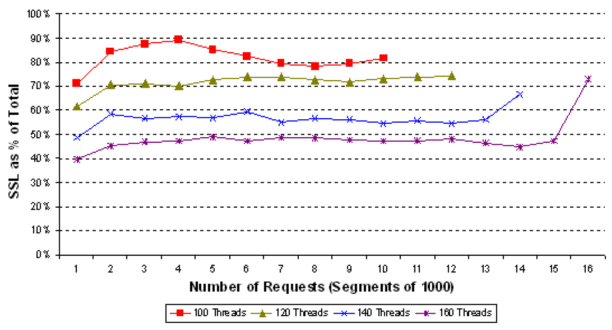

Figure 4: SSL handshake percentage of total time

Predictably, Figure 4 clearly shows that as the number of concurrent threads grows the percentage of the total time taken by the SSL handshake actually drops. This would not be the case if SSL was contributing to this extra processing time. In that case we would also see an increase in the percentage as well. This behaviour proves that adding mutual authentication to the periodical payment protocol does not have any unexpected side effects.

In fact, it is surprisingly easy to prove that the SSL handshake remains stable and consistent under increased server load. Plotting the averages using segments of 1000 requests on a time series chart (see Figure 5) reveals an unanticipated lack of variance between the different data series. Some points on this graph interlace showing how similar the times are despite increases in concurrent threads.

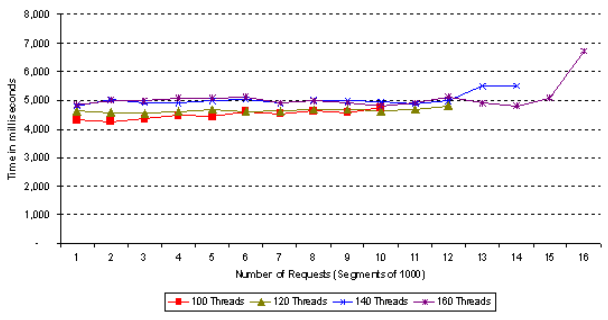

Figure 5: Average SSL handshake processing time

Having proven that SSL is not solely responsible for the sharp increase in processing times observed in Figure 2, we must find an alternative source of this variance. The only other significant component of the periodical payment framework that can potentially impact performance is the Apache Axis2 web services engine. This engine provides all of the “boilerplate” code required to implement a complete, standards compliant web services application. It performs all SOAP message handling logic and handles the entire web services communication. It is likely that this part of the application is causing the observed behaviour. However, it remains an exercise for the future to isolate exactly which component of this engine is causing performance issues.

The test results reveal that the impact of SSL with mutual authentication can be considered modest and reasonable. The benefits of using mutual authentication clearly outweigh the relatively small increase in performance figures. Considering that the behaviour of the application under load is stable we are not overly concerned with the initial results. Since SSL authentication can be implemented in hardware (see next section for further discussion) it is clear that this part of the application is unlikely to cause issues.

The other significant component of the application, the web services engine, is clearly underperforming. There is a significant amount of time spent within the web services communication layers. What is more disappointing is that as the load on the server increases the performance sharply degrades which is quite opposite to the SSL figures that displayed remarkable consistency. Fortunately, alternative implementations or technologies can be adopted to replace the Apache Axis2 web services engine if required.

5 Future Work

Detailed analysis of the periodical payment framework highlighted some issues with performance of both the SSL handshake and the web services communication components. While results look promising, a proper scalability test is required to ensure that this concept will scale to realistic load volumes. Before this can happen we need to isolate individual components of the web services stack to determine which part of the architecture is causing performance issues.

There are two related issues that need to be addressed regarding web services. The first is whether the web services concept is appropriate in the electronic commerce context. Web services provide interoperability and decoupling but do their extensive performance overhead justify their use? Our goal is to investigate alternative technologies that would provide a similar level of interoperability without the extra overhead.

As a second goal, we aim to evaluate alternative web services implementations. Focusing primarily on performance, the aim is to compare the Apache Axis2 engine to other frameworks offering the same features. It is quite likely that by switching to another web services provider performance gains can be achieved without any code changes.

Some performance improvements are also needed for the SSL handshake. There are various options at our disposal to make this part of the framework much faster. At present, Apache Tomcat servlet container running the payment gateway application is configured using JSSE. JSSE is a pure Java implementation of SSL, and as such it suffers performance issues. By integrating an alternative library developed using native APIs we can dramatically decrease the times taken by the SSL handshake. One such approach is to configure OpenSSL open source SSL library into Apache Tomcat using its Apache Portable Runtime (APR) mechanism.

Alternatively, an SSL hardware accelerator could be used to test the payment gateway in a production-like setting. A hardware accelerator is a special-purpose device designed to perform the SSL handshake in hardware thus delivering the best possible performance. These devices are commonly used by organisations with high Internet traffic that rely on SSL for authentication. While the cost of this device might prove prohibitive, it is none the less part of our agenda to investigate its use within the periodical payment framework.

In addition, the issue of customer credential storage should also be addressed. At the moment, our extension is loading a customer's keystore from a file. This is sufficient for testing but would hardly be acceptable in a final product.

In the past credential storage was an insurmountable issue. While smartcard technology has been around for many years, even most modern computers now are not equipped with smartcard readers needed to make this technology available for Internet use. This, however, is no longer a problem. Alternative technologies exist right now that make smartcard use a reality.

One such technology is USB keys. A USB key combines both the smartcard and a smartcard reader in a single device. The advantage of using USB keys is that USB is extremely popular with even relatively old computers supporting this standard. It is our goal for the future to integrate USB token support into our Firefox extension thus providing complete end-to-end implementation of periodical payments.

Another potentially promising technology that requires further research is using Subscriber Identity Module (SIM) cards commonly found in mobile phones as cryptographic tokens. SIM cards are essentially smartcards that are capable of storing customer cryptographic credentials. By integrating mobile phones into the periodical payment model we can leverage of the existing technology and infrastructure providing an alternative avenue for electronic commerce.

6 Conclusion

Periodical payment framework delivers a solution to an existing problem affecting users of the direct debit payment model. It solves issues of past implementations by leveraging of the currently available, standards compliant and industry supported technologies. At the same time it does not compromise on either security or performance.

It is clear that existing approaches to electronic commerce by industry leaders, such as Visa and MasterCard are not directly addressing this particular payment format. In fact, current attempts rely on customers' physical presence during payment transactions making delegation impossible. With the growing popularity of direct debit payments there is a clear need for a more secure way of conducting such payments over the Internet.

In this paper we have demonstrated that a periodical payment framework built using web services and secured using SSL with restricted proxy certificates is a viable solution to this problem. While using SSL with mutual authentication does impact performance, we have proven that this increase is insignificant when measured against the benefits it provides.

The use of web services has delivered mixed results. While we have benefited from its flexibility there are still questions regarding its performance. However, the data analysed so far is promising. Even under load, the payment gateway is performing admirably with linear performance degradation as the number of simultaneous requests grows.

There are numerous options still available to improve performance of both the SSL and web services components. As such, there is reason for optimism in the future of this approach.

7 Acknowledgements

We would like to thank Milutin Pribicevic for his help with analysing the test results. His insightful comments and suggestions were instrumental in preparing the performance analysis section of this paper.

8 References

Giampaolo, B., Lawrence, P., Massacci, F. (2002): The Verification of an Industrial Payment Protocol: The SET Purchase Phase. *Proc. 9th ACM Conference on Computer and Communications Security (CCCS2002)*, Washington, DC, USA, 12-20, ACM Press.

Giampaolo, B., Lawrence, P. and Massacci, F. (2003): Verifying the SET Registration Protocol. *IEEE Journal of Selected Areas in Communications*, **21(1)**:77-87

Goldman, G. (2007): Periodical payment model using restricted proxy certificates. *Proc. Thirtieth Australasian Computer Science Conference (ACSC2007)*, Ballart, Australia, **62**:131-139, ASC Inc.

Goldman, G. (2009): Periodical payment framework using restricted proxy certificates. *PhD Thesis*. Currently submitted for examination.

GPayments (2002): Visa 3-D Secure vs. MasterCard SPA: A comparison of online authentication standards, GPayments Pty Ltd.

http://www.gpayments.com/pdfs/GPayments_3-D_vs_SPA_Whitepaper.pdf. Accessed 27 July 2008.

Housley, R., Polk, W., Ford, W., Solo, D. (2002): Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. http://www.ietf.org/rfc/rfc3280.txt. Accessed 14 August 2009.

MasterCard, Visa (1997a): SET Secure Electronic Transaction Specification: Business Description. http://hostedfiles.110mb.com/2007/11/set-secure-electronic-transaction.html. Accessed 27 July 2008.

MasterCard, Visa (1997b): SET Secure Electronic Transaction Specification: Programmer's Guide. http://citeseer.ist.psu.edu/289529.html. Accessed 27 July 2008.

Paulson, L. (2003): Verifying the SET Protocol: Overview. *Formal Aspects of Security*. **2629**:4-14, Springer-Verlag, Berlin.

Stallings, W. (2006): Web Security: Secure Electronic Transaction. In *Cryptography and Network Security: Principles and Practice*. 4th ed., Prentice Hall.

Tuecke, S., Welch, V., Engert, D., Pearlman, L., Thompson, L. (2004): RFC3820: Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. http://www.ietf.org/rfc/rfc3820.txt. Accessed 4 August 2008.

Wolrath, C. (1998): SET: A market survey and a test implementation of SET technology. Accessed 27 July 2008. http://www.wolrath.com/set.html