Late Binding for the M-Pin protocol

Michael Scott

Chief Cryptographer
MIRACL Labs
mike.scott@miracl.com

Abstract. The M-Pin protocol has been proposed for use in a setting which uses multiple Trusted Authorities. One way to realise M-Pin is to use "early binding". With early binding the client secret shares issued by each authority are combined immediately after they are issued to each client, inside of the client process. Here we consider the possibilities of "late binding" whereby client secret shares are kept distinct. Conceptually with early binding all of the client secret shares are added to create a single secret, which is used to authenticate. Unless all shares are present and correct the authentication will not succeed. With late binding multiple individual secrets are issued as before, but each is used to authenticate separately. Unless all authentications succeed the overall authentication will fail. So in both cases the outcome is the same. However here we argue that late binding, at some extra cost, results in a more flexible system.

1 Introduction

The reader should refer to the original M-Pin paper for definitions.

A Trusted Authority (TA) can issue a client secret as sX, where s is their master secret, and $X = H(ID)$ is the client's identity hashed and mapped to an point on a specific elliptic curve, which we consider as an element in the group $\mathbb{G}_1$. Using the M-Pin protocol a client can now authenticate to an M-Pin server equipped with a server secret sQ, where Q is a fixed element in a different group $\mathbb{G}_2$.

Now a trivial way to distribute the client secret issuance functionality is to deploy n Distributed Trusted Authorities (DTA), each of which has its own master secret s_i. Client secrets can then be formed by simply adding all of these contributions, so $sX = s_1X + s_2X ... + s_nX$. Then proceed to authenticate to the server using this combined secret. The server has been issued with $sQ = s_1Q + s_2Q ... + s_nQ$. The benefit is obvious – an attacker must now capture all n components in order to reconstruct a client secret. Even if only one is missing, they get nothing. And clearly the master secret $s = s_1 + s_2 ... + s_n$ is itself similarly protected. So no more single-point-of-failure.

Observe now that an individual client secret share s_iX is indistinguishable in form from the full client secret sX. It is a scalar multiplication of the same element in $\mathbb{G}_1$. Therefore an alternative approach suggests itself. Instead of immediately adding all of the client secret shares, the client instead uses them all individually to authenticate to a server. Only if all these authentications succeed will the overall authentication be deemed a success by the server. The former approach we call "early binding", and the latter "late binding".

Now M-Pin also supports the concept of time permits. Here the identity used by the server to confirm authentication is formed as $X + Y$, where $Y = H(T|ID)$. Here T is a representation of time, perhaps the current day. Clearly the authentication will only succeed if the client has in their possession not only the client secret sX, but also a "time permit" sY.

With classic early binding M-Pin, each DTA issues client secrets shares, server secret shares, and time permits. With late binding, the client can now instead, by arrangement with the server, "do their own thing" with secret shares. They may decide to immediately add them (early binding), or to keep them all separate (late binding). Or they may decide to use some combination of both ideas. In particular DTAs which choose to issue time permits can be handled separately from those that do not. Therefore an immediate benefit is that not all DTAs need to issue time permits.

Another possibility is that PIN extraction might only be performed on one of the shares. Or that one DTA might issue secret shares from which a PIN will be extracted, and another might issue a share to be used with a fuzzy biometric, in which case the M-Pin error detection capability might be useful. Note that by keeping these shares separate, interactions between authentication "factors" can be removed. Also the server can now determine the exact cause of failure (bad PIN? bad biometric? wrong time permit?). In fact the possibilities are endless, and we will not attempt to enumerate them all here.

However clearly late binding requires more computation for both client and server, and an increase in bandwidth. So it should be used sparingly. Some simplifications are possible. Classic M-Pin is a 3-pass protocol, involving a client to server phase, a server issued challenge, and a client response. Since each authentication is independent, these passes can be combined so that the authentications take place in parallel, rather than serially. So the protocol remains 3-pass, although more data is transmitted in each pass. On the client and server sides more computation seems inevitable. However parallel processing may offset this.

2 Example configuration

Here we consider the simplest scenario which demonstrates the possibilities of late binding. We assume just two DTAs, one which supports time permits and one which does not, but from which a PIN will be extracted.

The first DTA with master secret share s_1 does not support time permits, and issues a secret $s_1 H(ID_a)$ to Alice and $s_1.Q$ to the server. The second DTA with master secret share s_2 and which will issue time permits, issues a secret $s_2 H(ID_a)$ to Alice and $s_2.Q$ to the server. In both cases $H(.)$ is a suitable hash function which hashes and maps an identity string to an element in the group $\mathbb{G}_1$.

In the protocol as described below, T_i represents the current time slot, and $s_2 H_T(T_i|ID_a)$ is the personalised "time permit", where $H_T(.)$ is another hash function. The extracted PIN number is α.

Alice - identity ID_a	**Server**
Generates random $x_1, x_2 < q$	Generates random $y_1, y_2 < q$
$A = H(ID_a)$	
$T = H_T(T_i\|ID_a)$	
$D = A + T$	
$U_1 = x_1 A$	
$U_2 = x_2 D$	
$ID_a, U_1, U_2 \quad \rightarrow$	
	$\leftarrow y_1, y_2$
$V_1 = -(x_1 + y_1)(((s_1 - \alpha)A + \alpha A)) \rightarrow$	$A = H(ID_a)$
$V_2 = -(x_2 + y_2)(s_2 A + s_2 T) \rightarrow$	$D = H(ID_a) + H_T(T_i\|ID_a)$
	$g_1 = e(V_1, Q).e(U_1 + y_1 A, s_1 Q)$
	$g_2 = e(V_2, Q).e(U_2 + y_2 D, s_2 Q)$
	if $g_1 \neq 1$ or $g_2 \neq 1$, reject the connection

Table 1. Late Binding M-Pin example

At first glance this configuration would be expected to double the computation and bandwidth. However a possible optimization would be for the server to calculate directly the product $g = g_1.g_2$, so $g = e(V_1, Q).e(U_1 + y_1 A, s_1 Q).e(V_2, Q).e(U_2 + y_2 D, s_2 Q)$, and to reject the connection if this were not equal to one. By bilinearity this can be simplified to $g = e(V_1 + V_2, Q).e(U_1 + y_1 A, s_1 Q).e(U_2 + y_2 D, s_2 Q)$. Products of pairings can be calculated very efficiently, and the expected overhead here would only be about 10%. However this would introduce the possibility of false positives, where $g_1.g_2 = 1$, but $g_1 \neq 1$ and $g_2 = 1/g_1$. But this would be very unlikely to happen in a real protocol run, and it is hard to see how an attacker might exploit this possibility without reversing the pairing function, which is believed to be hard. Another downside of this approach is that it would no longer be possible for the server to distinguish between a "bad time permit" and a "bad PIN" error.

Any error δ in the entered PIN can be calculated as before by solving the discrete logarithm problem $g_1 = e(U_1 + y_1 A, Q)^\delta$.