# SPR: Security Principles Assignment

Richard Ashworth

January 2013
Total Pages: 16

**Abstract**

This report forms my submission for the Security Principles module of the Software Engineering Programme run by the University of Oxford.

# Question 1

Since the 'DigiPound' system involves the transfer of money, the presence of threats is almost guaranteed. As software engineers, we must identify these threats and consider how potential vulnerabilities in the system might be exploited. We will then use the results of this analysis to help drive the security requirements that any serious implementation of the system must satisfy.

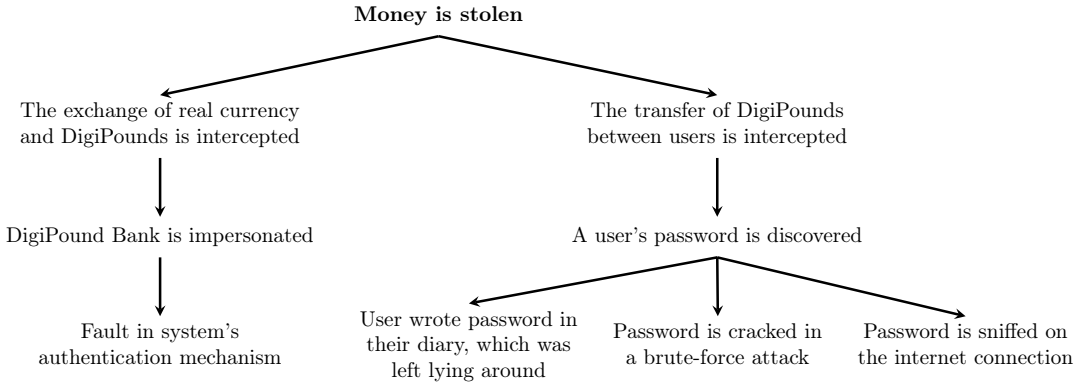
To establish the threats, vulnerabilities and security requirements of any system, we must first define what constitutes that system, and where its boundaries lie. The DigiPound system encompasses at least all of the following:

- Application software used to process the transactions
- The front-end(s) used to access DigiPound Bank and their development teams.
- The database and DBAs.
- The hardware on which the DigiPound Bank software runs.
- System administrators
- Users
- Devices used to handle DigiPounds and interact with DigiPound Bank.
- Public internet

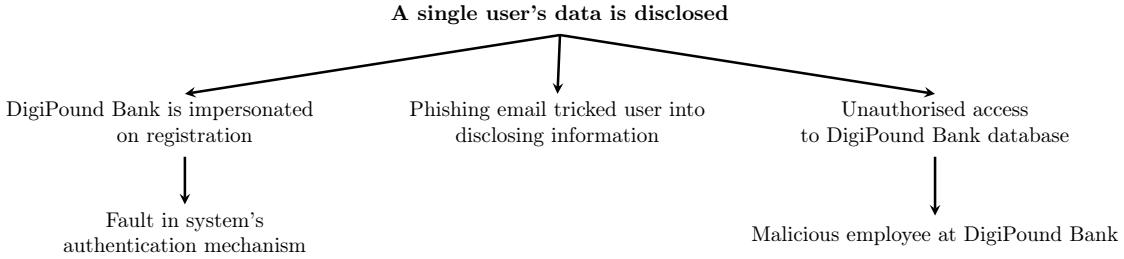
A security threat is made up of an Actor (which could be a human or a system), and an incentive. By considering how these components might combine, we can form a list of threats to the DigiPound system:

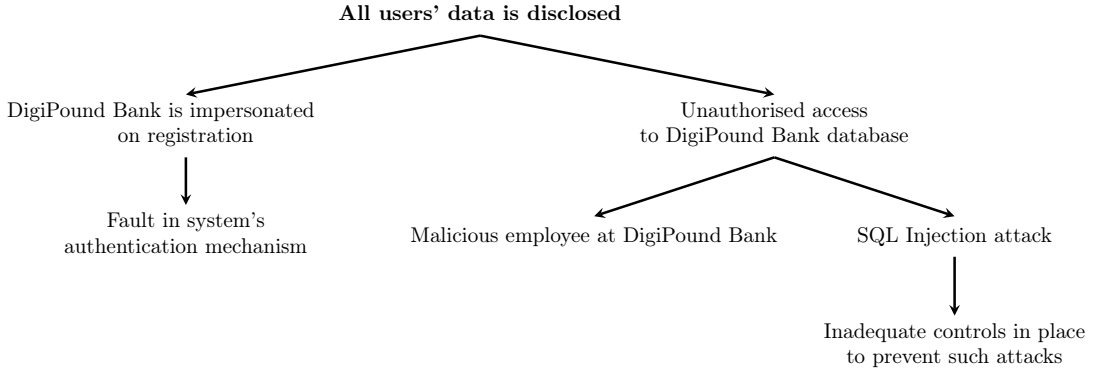
1. A thief wants to steal money that is transferred from the users to DigiPound Bank
2. A thief wants to steal DigiPounds that are transferred between the users.
3. A rival service wants to discredit the DigiPound Bank system.
4. A criminal group want to use the system for money-laundering.
5. The perpetrators of a spam campaign want to steal users' data.
6. A nosy friend of a particular user wants to access that user's data.

Once this has been constructed, we can perform a top-down analysis to identify the common roots of these threats. By tackling a threat at its root, we will also address those other threats which stem from that root. To do this, we begin at the *security event* that we wish to prevent (e.g. money being stolen), and decompose this into the properties of the system that may be involved in an attack. The tree below illustrates this decomposition for the threat that a thief wanting to steal money poses to the system:



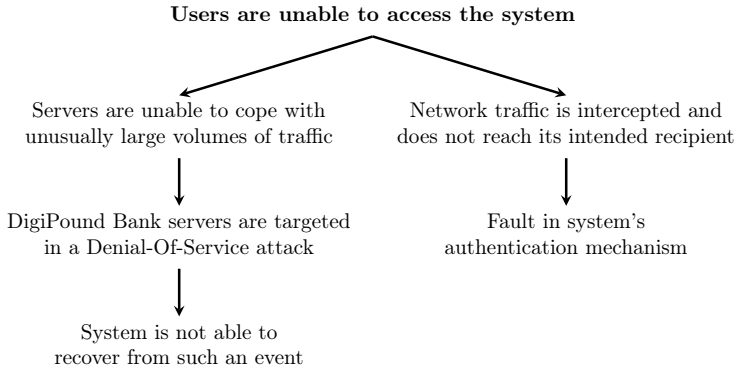

The roots of the threat derived above can be categorized according to their type. In our efforts to prevent the event (money being stolen), we must address each of these roots, since a system is only as secure as its weakest link. A fault in the system's authorisation mechanism could be exploited in an attack, as the user trusts that when they are buying DigiPounds, it is really DigiPound Bank that they are paying. Our analysis reveals that technical threats such as this are not the only ones we should concern ourselves with; the event could also originate from people - in this case, a user writing down their password. Given the significant role that people play in our definition of the system, we must acknowledge and respond to the threats that they pose. The combination of human and system threats is also present when we analyse the threat of data being disclosed - first for a single user, and then for all users in the system:



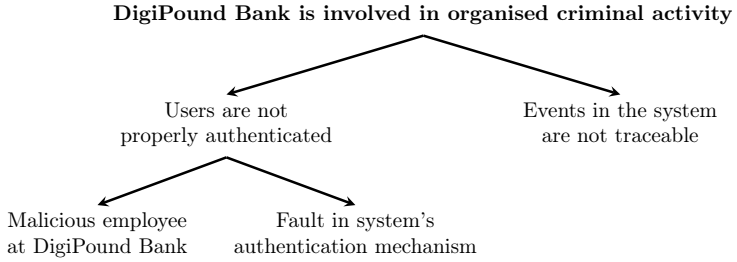



An important comparison can be drawn here between the disclosure of a single user's data and a *class attack* designed to extract every user's data. It is apparent that although the risk of all the users' data in the system being disclosed may be viewed as a much more serious threat, the root causes are almost identical; if it is possible to gain unauthorised access to one user's data, then it is likely that data for all users in the system has been compromised.

In the case of some threats, the cost of their prevention may be deemed to great to justify investment. In these cases, the system should at least be able to recover in the event of a security event, and all activity that occurred during the event should be traceable. The following two examples illustrate this need. The impact of the system being unavailable should be minimised by ensuring that it can recover in the case of an attack:




Similarly, while the system may not be able to prevent its involvement in certain criminal activities, it is important that where such activity occurs, it is recorded and can be traced.




Having discussed the threats faced by the DigiPound system, we see the emergence of common root causes and potential vulnerabilities. These then drive the security requirements that must be met by the system's implementation. Specific vulnerabilities will depend on the details of the implementation, but from the analysis above, we can identify several properties of the system that might be exploited in an attack. Preventing unauthorised access to the system depends on the security of the individual user's passwords. Passwords that are disclosed (inadvertently or otherwise) or can be guessed are both vulnerabilities here.

There are both business and technical requirements that might be put in place to address this vulnerability. With the requirement that passwords contain a minimum of eight alphanumeric characters, then there are $(26 + 10)^8$ possible passwords that a brute-force attack would need to consider in order to be guaranteed success. We might add further requirements to ensure that a brute-force attack remains the most efficient means of breaking the passwords: Users' names should be excluded, along with dictionary words. Although this slightly reduces the number of passwords that are accepted by the system, this business requirement defeats an attack based on guessing a combination that a naïve user might choose as their password.

We should bear in mind that we will not be able to identify all the threats to a system, and in any case, new ones are almost certain to emerge in the future. The security principles we use to design our system must therefore enable us to respond to these threats quickly and effectively when they are discovered.

# Question 2

Having identified a number of threats and vulnerabilities in the system, we must treat the requirements to handle these as first-class citizens in our design for the DigiPound. Security should be baked into the design and implementation of a system, not bolted on as an afterthought or delegated to an individual layer in the application. This is important since the abstractions used by the designers of the system may not align with those used by an attacker, who could look to exploit a vulnerability in the 'layer below' the implementation of security mechanisms.

We will aim to keep the DigiPound design as simple as possible; adding features only where they deliver value against the security requirements. Economy of mechanism is listed [SS75] as a design principle, and the easier it is to reason about a system, the more readily any shortcomings can be identified in its security. We can describe our design for the DigiPound as the identity of the DigiPound Bank ($DPB$), and a hash, encrypted with the DigiPound Bank's digital signature($S_{DPB}$), of a set containing the following elements:

- Unique Identification Code for the DigiPound, $id$
- Timestamp when the DigiPound was issued, $t$
- Amount of monetary units that the DigiPound is worth, $a$
- Expiry date of the DigiPound, $e$

We will use the following notation to represent the structure of a DigiPound:

$$\{DPB, \{id, t, a, e\}_{S_{DPB}}\}$$

The first requirement that our design for DigiPounds must satisfy states that users should not be able to generate DigiPounds arbitrarily. The simplest possible implementation of a DigiPound is an identification code, but this is not sufficient: DigiPounds can be created simply by generating these identification codes. This requirement therefore reflects a need to verify that all the DigiPounds in the system were issued by the DigiPound Bank. This would be achieved in the design of DigiPounds by incorporating a digital signature from the DigiPound Bank. Digital signatures use a form of asymmetric encryption in conjunction with a digital certificate so that if the decrypted hash of the plaintext matches a freshly generated hash of the message, we know that the message has not been tampered with, and it originates from the signer.

A limitation of this approach is that when the merchant, Bob, comes to verify the DigiPounds he has received from Alice with the DigiPound Bank, the Bank will know that those DigiPounds were issued to Alice, and will know what she has used them for. We could tackle this by using 'blind' signatures, in which the DigiPound Bank is not privy to the user's identification code that it is signing. However, since there is currently no requirement for this anonymity, we will not introduce any unnecessary complexity.

The second requirement we must address is to prevent users from spending more DigiPounds than they have available to them. The proposed design for the DigiPounds will handle this by with a unique identifier assigned to each DigiPound and present in the hash. On issuing a DigiPound, the DigiPound Bank will record its unique id against the user requesting it. After that DigiPound is spent, and the merchant (Bob) comes to verify it, a reconciliation will take place within the DigiPound Bank of the amount of DigiPounds that Alice has been issued and the amount being verified. If this verification is successful, Alice's 'balance' will be debited with the amount.

Although physical cash can be spent any number of times, we must preclude this transitivity in its electronic representation, since the DigiPounds must be verified after each purchase. This is captured in requirement (R3) of the system. Since the transactions between Alice and Bob take are not mediated by the DigiPound Bank, this requirement must be supported in the design of the DigiPounds themselves. In the case that a user tries to spend a DigiPound twice, although the user may be able to complete the purchasing of an item in step three, the transaction will be caught when the DigiPound is verified by the merchant. The DigiPound Bank can use their records to determine who Alice is from the identification number of the DigiPound, and in the case of persistent abuse, we can imagine that Alice might be barred from the system, or even prosecuted.

# Question 3

Authentication of the parties involved in a transaction plays a critical role in the DigiPound system. When users buy or verify DigiPounds with the DigiPound Bank, they need to be confident that they are not in fact interacting with an impersonator. A number of established protocols exist within the domain of Public Key Infrastructure, and in the first instance, we will look to these in our efforts to provide mutual authentication between Alice and the DigiPound Bank. The decision to reuse an existing protocol is grounded in a security principle. By leveraging established mechanisms that are open to scrutiny from the wider community, we will benefit from the discovery of flaws or enhancements, and since it will be clear what each of the involved parties knows, we will avoid the confusion of security with obscurity. The security of our implementation will be embedded in the strength of the users' keys, rather the details of some particular algorithm.

The Needham-Schroeder public-key protocol provides a means of mutual authentication between two agents, and can be described in its original form as follows:

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{K_B, B\}_{K_S^{-1}}$
3. $A \rightarrow B : \{N_A, A\}_{K_B}$
4. $B \rightarrow S : B, A$
5. $S \rightarrow B : \{K_A, A\}_{K_s^{-1}}$
6. $B \rightarrow A : \{N_A, N_B\}_{K_a}$
7. $A \rightarrow B : \{N_B\}_{K_B}$.

In the first step, Alice notifies the trusted key server $S$ that she intends to communicate with Bob. The key server then returns a message containing Bob's public key, $K_b$, and Bob's identity, encrypted with the server's private key. Alice can decrypt this message using the server's public key, and can then begin to send messages to Bob encrypted with $K_b$. She initiates this in step 3, and sends Bob her identity together with a nonce she has generated, $N_a$. On receiving this message (which *appears* to be from Alice), Bob notifies the key server that he wishes to send messages to Alice (step 4). The server replies to his request with Alice's private key and her identity in step 5, again, encrypted with the server's private key. The reason for including an identity as well as the key in these messages mean that the key for some other agent cannot be forwarded to Bob by an intruder who has intercepted the message; although the encryption on the message shows that it originated from the server, without the identity, we cannot be sure that this message is a response to the request for Alice's key. Now that both Alice and Bob are in possession of what they believe to be each other's public keys, Bob can respond to the nonce

challenge sent by Alice in step 3. Bob proves his identity to Alice by decrypting the nonce chosen by her (using his secret key), and responds to it with a message containing that nonce, and one of his own. In the final step of the protocol, Alice responds to this by decrypting the message sent in step 6, extracting Bob's nonce, and sending it to Bob encrypted with his public key.

The nonces are numbers generated randomly by Alice and Bob, and their use is restricted to a single run of the protocol. These numbers $N_a$ and $N_b$, sent in steps 3, 6 and 7, play an important role in this protocol; they provide authentication of the agents. Alice and Bob send each other their nonces in encrypted messages containing other information. The only way to return the nonce is for the recipient to decrypt the message using their secret key and extract it. If the agents receive the same nonce that they sent back, then they can be sure that the intended recipient has received their message, and they are not being replayed stale messages intercepted by a third party.

It was mentioned previously that we would favour security mechanisms that are publicly available, and we are able to gain some benefit immediately from having followed this principle. Since the Needham-Schroeder public key authentication protocol was published, a potential *man-in-the-middle* attack has been identified [Low95], and a fixed version of the protocol proposed. Full details of the attack are given in the solution to Question 6, but for completeness, the adapted protocol that would be used to achieve mutual authentication adds the identity of $B$ to the message sent in step 6:

$$6.\ B \rightarrow A : \{B, N_A, N_B\}_{K_a}$$

# Question 4

In establishing a protocol for transferring DigiPounds from one party to another, the first requirement is mutual authentication of the parties. This will be achieved using the adapted version of the Needham-Schroeder protocol described previously, reusing an established security mechanism. Once this mutual authentication has taken place, the transfer of DigiPounds can begin. A protocol to perform this is described below:

1. $A \rightarrow B : \{N_A, A\}_{PK_B}$
2. $B \rightarrow A : \{B, N_A, N_B\}_{PK_A}$
3. $A \rightarrow B : \{N_B\}_{PK_B}$
4. $A \rightarrow B : \{K_{AB}, \#\{N_A, K_{AB}\}_{S_A}\}_{PK_B}$
5. $B \rightarrow A : \{B, K_{AB}\}_{PK_A}$
6. $A \rightarrow B : \{item, N_A\}_{K_{AB}}$
7. $B \rightarrow A : \{tid, N_A\}_{K_{AB}}$
8. $A \rightarrow B : \{\#\{tid, a\}_{S_A}, \{DPB, \{id, t, a, e\}_{PK_{DPB}}\}\}_{K_{AB}}$

The first three steps are taken from our previous protocol, assuming that the public keys have already been exchanged in a previous run. In step 4, Alice notifies Bob (now that she is confident that she is indeed speaking to him) that she would like to purchase an item. Rather than continuing to use their public keys for encryption, Alice sends a weaker, though short-lived, shared key, $K_{AB}$. This will reduce the amount of processing power needed to encrypt and decrypt the messages, making the process cheaper, faster, and since an intruder seeking to participate in the transaction would need to crack the key while the session is still in flight, still sufficiently secure.

So that Bob can be assured that the proposed session key is fresh, and does actually originate from Alice, a nonce is included with this message, together with a signed hash of the message contents using Alice's private key. By comparing this signature with the message contents, Bob will be able to determine whether the key is from Alice and if it has been tampered with. Assuming Bob is satisfied with this, he proceeds by returning the nonce and key to Alice to confirm that he received it. This message is encrypted with Alice's public key to ensure that the key cannot be read or modified by a third party. With both Agents now securely in possession of the shared key, Alice begins the transfer of DigiPounds in step 6 by sending Bob a message containing the item that she wishes to purchase and a nonce. This message, and all those subsequent in this run of the protocol are encrypted with their newly established symmetric key.

Bob responds to the nonce challenge in step 7, and issues Alice a transaction identifier, *tid* that she can use to bind her purchase to a particular scenario. In this first iteration of the protocol, which we will seek to optimise later, Alice then sends the DigiPounds as individual messages to Bob, shown in our protocol as step 8. The messages contain a signed hash of the transaction id and the amount Alice claims she is sending, together with the DigiPound message itself. On decrypting this message (using their shared key), Bob will be able to generate a hash of the transaction id and amount himself, and verify that Alice's signature is authorisation for the correct purchase amount. Although he cannot directly inspect the contents of the DigiPound message, he will be able to forward it to the DigiPound Bank for verification and renumeration. By limiting our use of (relatively slow) public key encryption to the purposes of authentication, key exchange and generating signatures, we have made sure that the protocol is efficient, since encryption and decryption are significantly faster using the short-lived symmetric shared key.

# Question 5

On receiving the DigiPounds from Alice, Bob needs to verify that they are genuine and haven't already been spent before he can finalise the purchase. Verifying the DigiPounds is a function of the DigiPound Bank, and in order to submit them securely, Bob needs to use an appropriate protocol. The following protocol serves this purpose:

1. $B \rightarrow DBP : \{B, N_B\}_{PK_{DBP}}$
2. $DBP \rightarrow B : \{DBP, N_B, N_{DBP}\}_{PK_B}$
3. $B \rightarrow DBP : \{N_{DBP}\}_{PK_{DBP}}$
4. $B \rightarrow DBP : \{K_{BDBP}, \#\{N_B, K_{BDBP}\}_{S_B}\}_{PK_{DBP}}$
5. $DBP \rightarrow B : \{DBP, K_{BDBP}\}_{PK_B}$
6. $B \rightarrow DBP : \{a_B, \{DPB, \{id, t, a, e\}_{PK_{DPB}}\}\}_{K_{BDBP}}$
7. $DBP \rightarrow B : \{result \in \{\{valid, invalid\}\}_{K_{BDBP}}$

As with the protocols described previously in this report, mutual authentication and the exchange of a shared key are the goal of the first 5 steps, and here they function in exactly the same way as they have in the other protocols. In step 6, Bob sends details of how much he believes the amount of the signed DigiPound is, $a_B$, together with the DigiPound itself for verification to the DigiPound Bank. On receiving this, the DigiPound Bank will decrypt the DigiPound message and perform the validation. The DigiPound will be deemed valid if the following are all true:

- Decrypting the signed part of the message yields a correctly formatted DigiPound.
- The amount within the DigiPound matches the value $a_B$ in Bob's message.
- The expiry date $e$ has not elapsed.
- The unique identifier $id$ is found in a list of DigiPounds that have been issued.
- The unique identifier $id$ is not found in a list of DigiPounds that have already been spent.

Thus far, we have designed protocols which enable agents to send and verify DigiPounds, but they are not particularly efficient. In one purchase, Alice may need to send a large number of DigiPounds. The protocol currently requires that it be run in its entirety, or at least step 8 repeated for each DigiPound that is to be sent. For small amounts, this doesn't appear too great an issue, but if Alice is attempting to pay for an item worth several thousand DigiPounds, and the tokens she has are all denominated in small amounts, then running the protocols in their current form would put significant load on the system.

To reduce the number of messages that are exchanged between Alice and Bob, we could concatenate the $n$ DigiPounds in one transaction; sending a list of them, rather than each DigiPound individually. Since Alice has more spare compute cycles at her disposal, we can modify step 8 of the protocol so that she performs the concatenation of the DigiPounds. Step 8 of the protocol then becomes

$$8.\ A \rightarrow B : \{\#\{tid, a\}_{S_A}, \{DPB, \{id_i, t_i, a_i, e_i\}_{PK_{DPB}}\}_{1..n}\}_{K_{AB}}$$

where $\{DPB, \{id_i, t_i, a_i, e_i\}_{PK_{DPB}}\}$ is the $i$th DigiPound in a sequence from 1 to $n$. This optimisation reduces the number of messages required by a factor of $n$. The fewer messages that are encrypted using the weak shared key $K_{AB}$ makes it more difficult to compromise, and since the session key will be active for a shorter period of time, the adapted protocol also improves the security of the system.

# Question 6

Although we have endeavoured to implement security mechanisms through the design of our protocols, vulnerabilities still remain in the DigiPound Bank system. Had we not adapted the Needham-Schroeder public key authentication protocol, an attacker, Marvin, could attempt to steal the DigiPounds intended for Bob through a man-in-the-middle attack.

This attack targets a vulnerability in the exchange of messages between the agents participating in the protocol, and assumes that an intruder, $I$, has been eavesdropping on previous runs of the protocol and has recorded the encrypted messages that were sent. The attack is orchestrated through two instances of the protocol running simultaneously in order for the intruder to impersonate Alice during a session with Bob. In the following description of the attack, 'Msg $\alpha.n$' will denote the *nth* message sent in a session that Alice has initiated with the intruder, while 'Msg $\beta.n$' will denote the *nth* message from a session that the intruder has initiated with Bob:

$$\text{Msg } \alpha.1.\ A \rightarrow I : \{A, N_A\}_{K_I}$$

$$\text{Msg } \beta.1.\ I_A \rightarrow B : \{A, N_A\}_{K_B}$$

$$\text{Msg } \beta.2.\ B \rightarrow I_A : \{N_A, N_B\}_{K_A}$$

$$\text{Msg } \alpha.2.\ I \rightarrow A : \{N_A, N_B\}_{K_A}$$

$$\text{Msg } \alpha.3.\ A \rightarrow I : \{N_B\}_{K_I}$$

$$\text{Msg } \beta.3.\ I_A \rightarrow B : \{N_B\}_{K_B}.$$

In the attack presented above, Bob is left to conclude that Alice has initiated a session with them, and that they have subsequently authenticated successfully . As a *man-in-the-middle* attack, intruder Ivan (denoted '$I$', or '$I_A$' when using Alice's identity) has managed to impersonate Alice to Bob using information obtained by running the protocol simultaneously with Alice as themselves. Because the nonce exchange described in steps 3 and 6 of the original protocol does not include Bob's identity when he returns both nonces to Alice, she cannot be sure that it is in fact Bob's nonce that she has received. We can adapt the protocol to prevent this form of an attack by including Bob's identity with their nonces in his reply to Alice:

$$6.\ B \rightarrow A : \{\boldsymbol{B}, N_A, N_B\}_{K_A}$$

An attempt to run the attack would now fail, as Alice would be expecting Ivan's identity in Msg $\alpha.2$, since she believes (correctly) that she is communicating with Ivan:

$$\text{Msg } \alpha.1.\ A \rightarrow I : \{A, N_A\}_{K_I}$$

$$\text{Msg } \beta.1.\ I_A \rightarrow B : \{A, N_A\}_{K_B}$$

$$\text{Msg } \beta.2.\ B \rightarrow I_A : \{B, N_A, N_B\}_{K_A}$$

$$\text{Msg } \alpha.2.\ I \rightarrow A : \{\boldsymbol{B}, N_A, N_b\}_{K_a}.$$

Another possible attack that could be made on the DigiPound Bank system involves the restrictions on users from double-spending their currency. It was given as a system requirement that users should be prevented from spending the same DigiPound twice, and in the verification phase described in the solution to Question 5, we can see that the DigiPound Bank checks whether a DigiPound has already been spent before it can be accepted. Marvin, a malicious merchant, exploits this mechanism in the attack described below:

1. $M \rightarrow DBP : \{a_M, \{DPB, \{id, t, a, e\}_{PK_{DPB}}\}\}_{K_{MDBP}}$
2. $DBP \rightarrow M : \{valid\}_{K_{MDBP}}$
3. $M \rightarrow DBP : \{a_M, \{DPB, \{id, t, a, e\}_{PK_{DPB}}\}\}_{K_{MDBP}}$
4. $DBP \rightarrow M : \{invalid\}_{K_{MDBP}}$

Since Marvin submits the same DigiPound twice to the Bank's verification protocol, he is told that it is invalid on the second attempt. Assuming that the DigiPound has not already been spent, he is able to cash it in the first attempt, and then return the result of the second verification to the customer, claiming that their currency is invalid.

The protocols themselves form only a small part of the DigiPound Bank system, and there are a myriad of other areas through which it could be attacked. In our discussion of the way security could be implemented in the system, a key concept has been that of authentication; Alice only wants to send money to Bob when she is convinced that it is he who is receiving her messages. Authentication has been provided through the use of digital signatures, and on receiving them, trust has been awarded. The basis of this trust is grounded in Digital Certificates that associate an agent with their public key. Because a user trusts a key-signing authority, they implicitly trust all the certificates that they sign. However, when a key is compromised, the certificate should be revoked, so that it is not used as a basis of trust. This will typically happen by placing the certificate on a revocation list, but there is an assumption here that the actors in the system (browsers, smartphones, desktop clients etc.) will monitor this list and ensure that their certificates can be trusted. Where this does not happen, an attack surface appears. Alice and Bob could start a session using the DigiPound Bank PKI to establish a key, but if one of their public keys has been compromised, then unless the certificate is rejected, an intruder could access the session key and the encryption would be broken.

The devices used to perform the various transactions play an important role in the system's security. If Alice has malicious software (such as a keylogger) installed unwittingly on her smartphone, then despite the best efforts of the other players, all the information (including her passwords, the session keys, DigiPounds etc.) that she handles could be visible to a third party and facilitate an attack. The system must therefore decide carefully which platforms are to be supported, and how the security risks associated with each are to be managed.

Aside from the devices themselves, their connectivity to the network also provides a opportunity for an attacker to exploit. The protocols and encryption mechanisms involved in wireless networking, where cipher strength may be substituted for performance gains or lower power requirements are a prime target for attack, and they should be considered part of the DigiPound Bank system if such a means of accessing it is possible. A mis-configured firewall in the network could mask security flaws that were assumed to be fixed, and provide another vulnerability for an attacker to consider.

It has been assumed in our discussion of the protocols used in the DigiPound Bank system, that once a key has been established, the messages it is used to encrypt are secure. This is not

the case: they are as secure as the key and the method of encryption used. A strong key (at least 128-bit), together with an established encryption standard (such as AES) should be used in all channels of confidential communication. Where signatures are provided, they should again use an appropriate collision-resistant hashing algorithm. Since collisions have been identified in an MD5 implementation, Sha1 should be preferred instead. Care must be taken here, as some Certification Authorities still use the MD5 algorithm in issuing new certificates.

Perhaps the most important (and most difficult to manage) area to consider from a security perspective is the people that are involved with a system. The users do not form the only source of threats; indeed, the majority of security incidents occur from within a company, so security must be implemented with the same diligence on both sides of a firewall. In the case of the DigiPound Bank, we saw how a merchant could double-spend a DigiPound that they have received in order to steal it, and the same attack could be run just as easily by someone working at DigiBank. It is not just the malicious merchants or rogue employees that pose a threat. The internet is rife with tricks designed to obtain information without the user knowing who they are disclosing it to: email phishing campaigns would be a prime example. An administrator of the DigiPound Bank system could be tricked into disclosing their password to an attacker, or a merchant could be convinced to 'update' their details on a malicious third-party's website that he believes to be run by the DigiPound Bank. The appropriate use of cryptography and protocols is therefore important, but we must acknowledge the wider socio-technical nature of any public-key infrastructure in a discussion of its security.

# Bibliography

[Low95] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[SM07] S. Smith and J. Marchesini. *The Craft of System Security.* Pearson Education, 2007.

[SS75] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.