M.I.T. Laboratory for Computer Science

Request for Comments No. 266
December 19, 1984

Cryptographic Protocol for Trustable Match Making

by Robert W. Baldwin and Wayne C. Gramlich

1. Introduction

In a society that values personal privacy, the designers of computer services must carefully balance the trade-offs between anonymity and accountability. A service, such as a community bulletin board, that provides total anonymity is likely to be abused by people who wish to support illegal activities. On the other hand, a service, such as an electronic library, that keeps strictly authenticated records for accountability purposes is likely to present a major opportunity for the invasion of privacy. The problem is that authentication, which conflicts with anonymity, is the basis for accountability.

The work of David Chaum [Chaum 81] on digital pseudonyms suggests that technical mechanisms can be used to implement a middle ground. A system designer should have tools that allow some measure of anonymity without loosing the ability to control who is authorized to use the system and who is accountable for unusual activities. In search of tools that could be used to provide this middle ground, it is interesting to look at problems that explore the relationship between anonymity and authentication.

This memo presents a solution to the problem of implementing a trustable match maker. A match making program receives wishes from its users (e.g., **A** wants a date with **B**), and notifies them whenever a matching wish is found. For users to trust the service, they need to use it anonymously. However, when a match occurs, they want to be assured of the identity of the other party. That is, the users want to be authenticated to each other, but not to the match maker.

2. Scenario

Consider a possible college campus of the future. In this scenario all the students have easy access to modest processing, storage, and I/O resources. These resources are distributed in such a way that students can enforce access policies concerning these resources using simple physical security. Further, the campus has a communication network that provides access to much greater processing, storage, and I/O resources. These greater resources are the only mechanisms that students can conveniently use to share information with other students. Due to the sharing and the cost of these resources, they are administered by the college. The students do not control the policies that govern access to these shared resources.

The computers in this environment would be as much a part of a student's life as telephones are today. They would be used for both academic and social purposes. A classic social problem for shy students is arranging a date. If the students are shy, then they will not offer a date unless they are sure that the other person will accept. The problem is not in locating a person that one would be interested in, but in finding out whether a particular person would accept a date.

What the students might want to set up is a computerized third party that would listen to people's wishes, and notify both parties if matching interests are found. The requirement that both parties be notified of a match is necessary to discourage people from insincerely offering a date to discover another person's feelings.

Students will not use the computerized match making service if they do not trust it. Unfortunately, the resources that the students do trust (i.e., their local resources) does not provide the sharing necessary to implement the service. The service must use the less trusted but more powerful shared resources that are controlled by the college. To be able to trust the shared resource, they must use it anonymously, but the computation performed by the match maker is basically a comparison between identities, so it is not obvious how this problem could be solved.

3. Goals and resources of the attacker

To understand the purpose of each step in the protocol used to solve this problem, it helps to have a list of the goals and resources of the people trying to break this system. The general aim of an attacker of this system is to find out other people's wishes without participating in a match. This includes learning that **A** is interested in **B**, or that **A** and **B** have successfully matched.

A wide range of resources is assumed to be available to an attacker. All information that a process stores or transmits is available. Further, the attacker can send and receive messages to any host and can perform all the publicly available cryptographic transforms. A list of these resources is given below.

* Record messages.

* Read match maker's storage.

* Perform public transformations.

* Send messages.

* Write match maker's storage.

* Subpoena M's help.

One resource deserves special attention. The attacker can subpoena the match maker's help. The main difficulty in solving the match making problem discussed in this paper is that the computer used to run the service is not under the control of a trusted party. Any static secrets known to the match maker program (e.g., keys compiled into the program) can be made known to the system administrators. It is assumed that the program cannot be modified by the system administrators, or at least that the users can tell when it has been modified.

The major resource not on this list is traffic tracing. We assume that it is not feasible to trace the source of a message back to the user (process) that originated it. Attacks based on traffic tracing can be thwarted by an anonymous message forwarding service. Another resource not included in the list is the ability to read or write the storage that belongs to users of this system. Any information that the users need to remember can be stored by their local resources, which are trustable in this scenario.

4. A Solution

Before presenting a protocol that can be used to solve the match making problem, it helps to see how the problem could be solved if everyone could be trusted. Following that, the general solution will be described and discussed. An argument for the correctness of this solution is presented in section 5.

4.1. Match making in an ideal world

The protocol for the match making service will be expressed in terms of two users, **A** and **B**, and a match maker, **M**. According to the scenario, the match maker, **M** is a program running on some computer administered by the college. The object of the protocol is to allow **A** to find out whether **B** would accept a date, and vice-versa. This match making service is not a computerized question and answer comparison. Rather, it provides a third party that will listen to user's wishes and let them find out when a matching wish has been heard.

In a world where everyone is trusted (i.e., people tell the truth and behave as expected) the protocol listed below could be used to solve the match making problem. This is not the only possible protocol to perform the match making in such a friendly environment, but its details match the details of the final protocol presented in section 4.2. After listing the simple protocol, these details will be discussed.

About notation: The protocol is expressed in terms of two people, **A** and **B** using the match making service provided by **M**. The symbol **||** represents the string concatenation operator.

1. **A→M: A||B, A**
   M stores: A||B, {A}

2. **M→A: NoMatchYet**

3. **B→M: A||B, B**
   M stores: A||B, {A, B}

4. **M→B: Match**

5. **A→M: A||B, A**

6. **M→A: Match**

A expresses interest in **B** by sending the string **A||B** to the match maker. The string can be viewed as a label that identifies **A**'s wish. Both **A** and **B** use the same label for the wish, so part of the protocol is an agreement about the canonical ordering of the two identifiers in a wish label. To allow **M** to distinguish between a wish coming from **A** and a wish coming from **B**, the first and third steps also include an identifier. In the case of the simple protocol, the identifier is the name of the person sending the message. The match maker will only report a match if it receives two messages with the same wish label, but different identifiers.

In the first two steps of this protocol the match maker learns that **A** is interested in **B**, and tells **A** that it has not heard a matching wish. Some time late the next two steps are executed. Those steps are similar to the first two, but the match maker tells **B** that a matching interest has been found. After some delay, perhaps days later, **A** repeats the wish, and this time finds out that a match has occurred.

An alternative to steps five and six, is to have the match maker explicitly notify **A** when the matching wish is received from **B**. That alternative is not acceptable because in the full solution the match maker will not know the identities of **A** and **B**.

This simple protocol can be attacked in many ways and a few of them are worth special mention. One attack tries to get around the joint notification requirement. It works by waiting until the match maker is about to delete old wishes before putting in a wish. The person submitting the late wish will find out about the match, but the other party is unlikely to check again before the wish is deleted. A simple way to avoid that is to make wishes valid for one week and to delete them after two weeks. At the end of the first week the status of a wish is committed to be either matched or unmatched. The remaining week before it is deleted gives both parties a chance to check its status. This solution is used in the full match making protocol.

If one of the parties, say **B**, can read **M**'s storage, then **B** can learn of **A**'s wishes, simply by reading **M**'s storage. This is an example of attacks based on examining storage and attacks based on not completing the protocol. A further example is that in step 3, **B** could send the message **A||B, C**. This message would cause **A** to be confused in step 6, though it would allow **B** to find out that **A** is interested. In general, a user can choose to incorrectly complete the protocol.

The last attack to examine is masquerading as **M**. Such an attack allows a person to learn all the secrets that are revealed to **M**. This attack will not be very useful in the final protocol, because the goal of that protocol is to keep the match maker in the dark. A person masquerading as **M** could not learn much more than **M** could, because the only knowledge they have that the match maker doesn't, is their own secret keys.

Anyone masquerading as **M** is constrained to correctly carry out the match maker's end of the protocol. The reason for this is that any user can invent a pair of people and play both halves of a match. With these invented matches, the users can check the behavior of the match maker. This

is an important feature. The users do not have to trust that the match maker behaves, they can check its behavior explicitly.

4.2. Trustable match making

An essential tool used to solve this problem is introducing fake transactions that are used to cover the information that is hard to hide using cryptography. The protocol listed below is carried out both by sincere users and by automated jokers. The jokers generate fake transactions that a third party cannot distinguished from real transactions. Basically, the jokers are constantly framing people by making the match maker's records look like random pairs of people requested and received a match. An important result of this random framing is that no stigma can be attached to the existence of records that indicate that someone used the service.

About notation: Following the standard convention, the symbol `E` denotes encryption and `D` denotes decryption. These symbols are used for both public and conventional encryption. The two cases are distinguished by a prefix on the key. For example, E_{CKI} denotes conventional encryption using the key `CKI`, and E_{PKA} denotes encryption under `A`'s public key. The protocol uses several randomly generated conventional keys represented as `CKRA`, `CKRB`, `CKI`, and `CKJ`. The symbol, $\circ$, denotes functional composition.

The protocol listed below in discussed in section 4.3.

Registration

1. `A` registers a public key `PKA` with `M`.
2. `B` registers a public key `PKB` with `M`.
3. `M` registers a public key `PKM`, selects a secret conventional key `CKM`.[1]

[1] That is, `M` keeps two secrets `SKM` and `CKM`. See section 6 for a discussion of this.

Bid

The effect of the Bid and Lookup stages is for **A** and **B** to exchange two one-time keys, **CKRA** and **CKRB**, without revealing them to **M** or anyone else.

1. **A**→**M**: **B**

2. **M**→**A**: **PKB**

3. **A**→**M**: $E_{PKM}(CKI)$, $E_{CKI}(A||B)$, $E_{PKB}(CKRA)$
   M stores: $E_{CKM}(A||B)$, $\{E_{CKM} \circ E_{PKB}(CKRA)\}$
   A stores: $E_{PKB}(CKRA)$, **CKRA**, **A**||**B**

B performs three similar steps:

4. **B**→**M**: **A**

5. **M**→**B**: **PKA**

6. **B**→**M**: $E_{PKM}(CKJ)$, $E_{CKJ}(A||B)$, $E_{PKA}(CKRB)$
   M stores: $E_{CKM}(A||B)$, $\{E_{CKM} \circ E_{PKB}(CKRA), E_{CKM} \circ E_{PKA}(CKRB)\}$
   B stores: $E_{PKB}(CKRB)$, **CKRB**, **A**||**B**

Lookup

1. **A**→**M**: **A**||**B**
   M computes: $\{D_{CKM} \circ E_{CKM} \circ E_{PKB}(CKRA), D_{CKM} \circ E_{CKM} \circ E_{PKA}(CKRB)\}$

2. **M**→**A**: $\{E_{PKB}(CKRA), E_{PKA}(CKRB)\}$

3. **A** computes: **CKRB** = $D_{SKA} \circ E_{PKA}(CKRB)$
   A stored the pair $E_{PKB}(CKRA)$, **CKRA** in step 3 of Bid, so **A** can recall **CKRA**.

B performs three similar steps:

4. **B**→**M**: **A**||**B**
   M computes: $\{D_{CKM} \circ E_{CKM} \circ E_{PKB}(CKRA), D_{CKM} \circ E_{CKM} \circ E_{PKA}(CKRB)\}$

5. **M**→**B**: $\{E_{PKB}(CKRA), E_{PKA}(CKRB)\}$

6. **B** computes: **CKRA** = $D_{SKB} \circ E_{PKB}(CKRA)$
   B stored the pair $E_{PKA}(CKRB)$, **CKRB** in step 3 of Bid, so **B** can recall **CKRB**.

Check

There is now a value, $E_{CKRA} \circ E_{CKRB}(A||B)$, that anonymously identifies the wish A||B. This value is submitted to M to check for a match. The match step is an atomic store and lookup operation. The values N_A and N_B are randomly chosen by A and B respectively.

1. $A \rightarrow M$: $E_{PKM}(E_{CKRA} \circ E_{CKRB}(A||B)||N_A)$
   A stores: $E_{PKB}(CKRA)$, CKRA, A||B, N_A
   M computes: $D_{SKM} \circ E_{PKM}(E_{CKRA} \circ E_{CKRB}(A||B)||N_A)$
   M stores: $E_{CKM} \circ E_{CKRA} \circ E_{CKRB}(A||B)$, $\{E_{CKM}(N_A)\}$

2. $M \rightarrow A$: NoMatchYet

3. $B \rightarrow M$: $E_{PKM}(E_{CKRA} \circ E_{CKRB}(A||B)||N_B)$
   B stores: $E_{PKB}(CKRB)$, CKRB, A||B, N_B
   M computes: $D_{SKM} \circ E_{PKM}(E_{CKRA} \circ E_{CKRB}(A||B)||N_B)$
   M stores: $E_{CKM} \circ E_{CKRA} \circ E_{CKRB}(A||B)$, $\{E_{CKM}(N_A), E_{CKM}(N_B)\}$

4. $M \rightarrow B$: Match

5.

6. A recalls N_A from storage, repeats step one, and learns about the match.

4.3. Discussion

The first stage of the protocol, Registration, records the binding between users and public keys. This binding could be maintained by another service, such as a mail server, but it is explicitly included in the protocol to point out where authentication is done. The degree to which users are authenticated when they register public keys limits the degree to which users of the match making service can be sure who they are dealing with. The pitfall in using another service's key server is that the match making protocol might be compromised by facilities provided in other protocols that operate on a user's public and secret keys. Section 5 argues that the match making protocol does not force people to use their secret keys in compromising ways.

The check stage of this protocol is similar to the simple protocol described in section 4.1. In the simple protocol the wish label is A||B, whereas in the full solution it is $E_{CKRA} \circ E_{CKRB}(A||B)$. The wish label in the later case includes sufficient information for the users to authenticate each other, but not enough information for the match maker to know who was involved in the match.

In the simple protocol a flag, either A or B, is sent along with the wish label. The match

maker looks at the flag to see whether the message is a repeat of a wish that it has already seen. In the full solution, the two random values N_A and N_B serve the same purpose. It is assumed that bit twiddling the messages in steps one and three of the Check stage cannot produce a message that has the same wish label, but a different random number. Otherwise such a bit twiddled message could be used to make **A** think that a match occurred when it had not. Such a failure is serious given the scenario of this paper. However, the users of the service can detect a bit twiddling attack if it is done frequently. They can make up fake wishes, and notice that they are fulfilled when they shouldn't be.

One ambiguous point in the protocol is what happens if matching interests are not found? In keeping with the strategy of adding noise to hide information that is hard to hide with cryptography, the match maker invents fake matches. Specifically, in steps three and six of the Bid stage the match maker stores a random number of fake values along with the real value received from **A** or **B**. The users of the service must complete the Check stage to see if any of them are real matches.

5. Correctness Argument

The goals of the protocol are to allow users to find out when they have matching wishes, and to prevent everyone who is not involved with the match from learning about it. To achieve the first goal, the parties involved in a match must be able to distinguish fake matches from real ones. A modified challenge-response protocol is used to do that. The goal of anonymity is achieved by using public key cryptography to pass information from **A** to **B** (and vice-versa) through **M** without **M** finding out the information.

The heart of the authentication between **A** and **B** is that **B** must decipher a value, $E_{PKB}(CKRA)$, to find a value that **A** picked randomly. Symmetrically, **A** must decipher $E_{PKA}(CKRB)$ to find, **CKRB**, which **B** chose randomly. These deciphered values, **CKRA** and **CKRB** are then used to compute a wish label, $E_{CKRA} \circ E_{CKRB}(A||B)$, which is sent to the match maker. A match is noted if **M** receives two different messages that both contain the wish label. As is argued below, being able to decipher a randomly chosen value that has been encrypted under **B**'s key, is reasonable proof that **B** was the other person involved with the match.

An important property of this challenge response protocol is that a person who correctly

carries out one end of the protocol can tell if the other end is being performed correctly. However, a joker can simulate both ends of the protocol, so jokers can be used to generate fake transactions that are indistinguishable from real ones. Specifically, a joker can create matching pairs of x and $E_{PKA}(x)$ for A or any other person. Using two such pairs, a joker can generate a wish label and carry out the full protocol. The protocol is strong enough for one end to authenticate the other, but not strong enough for a third party to be sure who was involved with the match.

The fake transactions also allow the users of the service to check up on the behavior of the match maker. Even if there is some way that an attacker could make a person think that a match occurred when none did, users could detect the presence of such an attacker by submitting fake match requests that they know should not be matched.

In general, the deciphering process requires knowing a secret key and knowing a person's secret key is treated as proof that one is that person. In this protocol a single random value is enciphered, and being able to decipher it is used as proof of identity. The question that must be answered is whether there is some way an attacker that does not know the secret key can decipher the given value, or otherwise compute the wish label for a match. If so, then an attacker could make A think a match happened when none occurred. That would be a failure to meet the authentication goal.

Starting at the Bid stage of the protocol, the pieces of the key value (CKRA and CKRB) are encrypted under B's and A's public keys. To learn the value of these pieces, an attacker must somehow apply D_{SKB} and D_{SKA}. Those functions only are applied in steps three and six of the Lookup stage, and the resulting values are never stored or transmitted. Thus, an attacker cannot find CKRA or CKRB. However, to cause a fake match, the attacker only needs to computer the wish label, which might be possible without knowing CKRA and CKRB.

Is there a way that an attacker can learn the wish label without knowing CKRA and CKRB? The wish label is transmitted in steps one and three of the Check stage, but they are encrypted under M's public key. The impact of the attacker knowing M's secret key is discussed in section 6, so it will not be considered here. The effect of this encryption and the encryption used by M to store the wish label is that an attacker cannot directly learn the value of any wish label. However, an attacker that is trying to create a fake match does not need to submit the fake match himself. It

would be enough to have someone else submit it to M. This sort of indirect attack is prevented by the nature of the wish label. The wish label, $E_{CKRA} \circ E_{CKRB}(A||B)$, depends on two random values, one chosen by A and one chosen by B, and on the plain-text wish, A||B. An attacker cannot force a user to generate a specific wish label. A real user will always randomize part of the value. Similarly, it is not possible to take a wish label that was used for one purpose and reuse it for another purpose.

The other goal of the protocol is ensuring that the wishes are processed anonymously and in secret. Neither M nor B should be able to tell when A sincerely offers a date with B. To keep M in the dark, jokers are constantly generating fake wishes. B is kept from knowing about A's wishes, fake or sincere, by using a two level encryption in step three of the Bid stage. Notice that one level of encryption such as

$A \rightarrow M: \quad E_{PKM}(A||B),\ E_{PKB}(CKRA)$

is not good enough because B can compute $E_{PKM}(A||B)$ and monitor the network for that value. This is a dictionary attack. Similarly, to keep B from learning about a wish by examining M's storage, M encrypts with CKM all information from step three of the Bid stage before storing it. The key CKM is not known to any of the users, so attackers cannot directly build up a dictionary. To prevent an attacker from indirectly building a dictionary by sending wishes to M and looking for new entries in M's storage, entries must be randomly changed and written in large groups. Further, any information that would allow an attacker to identify a particular entry must be hidden using cryptography. Specifically, $E_{PKB}(E_{CKRA})$ must be encrypted because it is a value that a user can recognize in M's storage, and thus it could be used to identify the value $E_{CKM}(A||B)$.

6. Programs Keeping Secrets

To run this protocol, the match maker program must generate and keep secret two keys, CKM and SKM. In this scenario the administrators of the computer used to run the program can find out these keys, so it is worth pointing out what they can deduce if they know these keys.

Using those keys, the administrators can find out anything that is known to M. However, as was argued in the last section, M cannot know who is using the service or which wishes are sincere. Without further information, the administrators cannot learn any useful information. To learn more they would have to know the secret keys of some of the users. Using just CKM and SKM, the most damage the administrators could cause is incorrect matches either by writing M's storage or by decrypting the messages sent during the Check stage to determine the wish labels.

If an administrator who knew CKM and SKM conspired with a user, then only a small number of the wishes would be compromised. Say B has been told M's keys, then B could find out about those wishes that involved B without completing the protocol (i.e., without allowing joint notification). B can do that by looking at M's storage (or monitoring the network) to find all the fake or real bids that involve B. For example, after A completes step three of the Bid stage B can apply D_{CKM} to M's storage to learn of A possibly real wish. To validate that wish, B would complete all the steps of the protocol up to, but not including the Check stage. Instead of performing the Check stage, B would look at M's storage to see if the other person had completed the Check stage. If so, the match was sincere. This attack can only be performed if the attacker knows a person's secret key. It cannot be used to find out about wishes that do not involve the person who's secret key is known. Thus, even if M's keys are compromised, the impact is limited to the few people who know the compromised keys. Hence, the security degrades gracefully.

7. Conclusions

In search of tools that can be used to achieve both anonymity and accountability this paper has posed and solved a problem that has conflicting goals of authentication and anonymity. Authentication is often the basis for accountability, so this problem is relevant to the trade-off between anonymity and accountability. When a user is told by the match maker that a matching wish has been heard, the user should be able to know that matching wish came from the correct person. That is, the service must meet the goal of allowing users to authenticate each other. However, the users do not fully trust the computer that runs the match making service. They want to use the service anonymously, so the match maker cannot figure out their true wishes.

To provide authentication, the solution uses a challenge-response protocol based on public key cryptography. A public key system is essential because it allows two users to pass information through a third party without the third party learning the information, and without needing to interact with each other in real time. The particular challenge-response protocol was chosen because it is strong enough for one side, which is correctly following the protocol, to authenticate the other, but not strong enough for a third party to record the exchange and verify who was involved. In fact, a joker can perform both side of the protocol to create a fake transaction.

Fake transactions serve two important functions in the solution. Like fake messages generated to thwart traffic analysis, these transactions hide information that is hard to hide using

cryptography. For example, the match maker is told during the Bid stage that one user is interested in another, but the match maker cannot tell if the request is real or generated by a joker. The other feature of the fake transactions is that they allow the users to check up on the match maker's behavior. Users do not have to trust the match maker to carry out its half of the protocol. They can generate fake transactions and test whether it is behaving correctly.

References

[Chaum 81] David L. Chaum.
Untraceable Electronic Mail, Return Address, and Digital Pseudonyms.
Communications of the ACM 24(2):84-88, February, 1981.