Receiver Anonymity via Incomparable Public Keys

Brent R. Waters   Edward W. Felten   Amit Sahai
Department of Computer Science
Princeton University
{bwaters,felten,sahai}@cs.princeton.edu

ABSTRACT

We describe a new method for protecting the anonymity of message receivers in an untrusted network. Surprisingly, existing methods fail to provide the required level of anonymity for receivers (although those methods do protect sender anonymity). Our method relies on the use of multicast, along with a novel cryptographic primitive that we call an Incomparable Public Key cryptosystem, which allows a receiver to efficiently create many anonymous "identities" for itself without divulging that these separate "identities" actually refer to the same receiver, and without increasing the receiver's workload as the number of identities increases. We describe the details of our method, along with a prototype implementation.

Categories and Subject Descriptors

E.3 [**Data**]: [Data Encryption]

General Terms

Security

Keywords

Anonymity, PGP, Public Key Cryptography, Privacy

1. INTRODUCTION

Anonymity is a desirable property in many communication systems. Although several good methods exist to protect the anonymity of message *senders*, there appears to be no existing method that fully protects *receiver* anonymity. We address this problem by presenting a system that protects receiver anonymity for a wide range of message-passing applications.

Prior research has focused on how receiver anonymity can be compromised because of the contents of a message or because of how a message is routed. However, receiver anonymity can also be compromised by the actual contents of the public key used to encrypt messages. If several senders share the same public key to encrypt messages to be sent for an anonymous receiver, then they can infer that they are indeed sending to the same receiver. They can then aggregate the information they each have on that receiver to further compromise his anonymity. To achieve receiver anonymity a receiver must be able to create several truly anonymous identities that will allow for a sender to both encrypt and route messages to him. These identities must be incomparable in the sense that the two cannot be identified as belonging to the same individual.

CCS'03, October 27–30, 2003, Washington, DC, USA.

1.1 Requirements

Our goal is to provide a way for senders to transmit messages to receivers, but without anyone — including the senders — being able to determine which message is destined for which receiver, or even being able to determine whether any two messages are destined for the same receiver. We want to keep this information secret not only from outsiders, but also from message senders, since the sender is often the very person from whom the receiver most wants to conceal his identity.

We assume that adversaries can see both the contents of messages and where those messages are routed.

We define three requirements that must be met for receiver anonymity to be realized. The first requirement is that if any conspiracy of senders and eavesdroppers is asked to determine the receiver of a particular message, they can do no better than random guessing.

In practice, we cannot prevent receivers from replying to messages, or from divulging information about their identities in these replies. Whenever a receiver replies to a message, some small amount of information about that receiver's identity will probably leak. (For example, Rao and Rohatgi [16] describe how a surprisingly large amount of information about authorship can be extracted from text documents.) If the receiver is providing some service to the sender or vice versa, some leakage of this type may be a necessary consequence of that service. Since we cannot prevent this type of leakage, our goal is to make sure that the adversary cannot get any useful information other than this.

This type of information leakage motivates the second requirement of anonymity: each receiver must be able to create a large number of anonymous identities, such that any message sent to any of these identities will go to that receiver, but nobody else will be able to tell that those anonymous identities correspond to the same receiver. This requirement is important in an environment where senders can learn a little information about each receiver; by preventing the senders from learning that they are talking to the same receiver, we prevent them from aggregating the information they have about that receiver.

This second requirement implies that anonymous identities must be *incomparable*, so that an adversary who sees two or more identities cannot tell whether they correspond to the same receiver. Practicality also requires that many messages can be sent to the same anonymous identity without compromising the receiver's anonymity.

The third requirement is that the solution be reasonably efficient. Some of the obvious approaches to the problem fail for efficiency reasons. For example, schemes that use a separate private key for each potential sender are too inefficient, as they require the receiver to try each of the possible private keys when a message arrives[1].

As described in Section 2, none of the existing systems for receiver anonymity can fully meet these criteria.

1.2 Our Solution

Our solution meets all of the requirements mentioned above. We prove its cryptographic properties, under the standard assumption that the Decisional Diffie-Hellman problem is hard. We also describe an implementation.

In our solution, each message is sent to a multicast group. All members of the multicast group try to decrypt the message, but only one of them will succeed. The other members of the group, having failed to decrypt the message, simply ignore it. The adversary can, of course, tell that the intended receiver is one of the members of the multicast group, so this provides anonymity only within the multicast group. However, if the multicast group is large enough this will be acceptable in practice. (This aspect of our solution is not novel, and the remainder of our solution can be used with routing methods other than multicast.)

The cryptographic component of our solution involves the use of a new class of asymmetric key cryptosystem, which we call an Incomparable Public Key cryptosystem. In an Incomparable Public Key scheme there exist many unique public keys that can each be used to encrypt data in such a way that a single secret key can be used for decryption. We call two public keys "equivalent" if they correspond to the same secret key and "non-equivalent" if they do not. The crucial property of such a cryptosystem is that an adversary, given two public keys, cannot tell whether they are equivalent.

Using an Incomparable Public Key scheme a receiver can construct an anonymous identity as the pair of a multicast address and an Incomparable Public Key. To construct another anonymous identity the receiver just uses the same multicast address with an equivalent, but unique public key. The identity is truly anonymous since colluding senders cannot distinguish two equivalent public keys from two non-equivalent public keys. A sender can repeatedly encrypt and send messages to the multicast address without requiring any response from the receiver. The receiver can identify and read messages efficiently as it only needs to perform one decryption for each message sent to the multicast address.

[1]Of course, one could try putting a marker on each message to identify which private key to use for decrypting that message, but these markers would leak identity information in violation of the anonymity requirements. The details of these and other design alternatives are discussed at greater length in Section 4.1.

Our solution allows a receiver to create and use as many or as few anonymous identities as desired. For maximum anonymity, a receiver might want to give a separate anonymous identity to each sender. Indeed, a receiver might want to give multiple anonymous identities to the same sender, if the receiver is carrying on multiple independent conversations with that sender. Alternatively, a receiver may choose to give the same anonymous identities to a set of senders, to enable that identity to develop a reputation among those senders, while using separate anonymous identities with other senders. We expect that different receivers will have different policies for managing anonymous identities, and our solution does not constrain this policy choice.

1.3 Practical Motivation

Before we describe the details of Incomparable Public Keys we briefly discuss situations that motivate the use of Incomparable Public Keys in practice. We believe that there are two general classes of situations for which Incomparable Public Keys are useful. The first is when the receiver wishes to communicate with another distinct party from whom he wishes to remain anonymous, either because he does not want that party to know his identity, or because he does not trust that party to keep his identity secret. The second situation occurs when the receiver wishes to communicate with a device that he either controls or trusts, but the receiver is concerned with maintaining his anonymity in the event that the device becomes compromised.

The first situation is the one most commonly explored in anonymous communication systems. Examples include anonymous web-browsing and e-mail. The anonymous identity of the receiver, which includes an Incomparable Public Key, can be delivered anonymously (see related work in Section 2) to any sender that is willing to communicate with an anonymous receiver. The important aspect of this situation is that the receiver does not trust the other party to begin with, but both the parties still observe a mutual benefit in communicating anonymously.

In the second situation the receiver might trust or even control a set of devices (or the parties behind them), but wishes to forward protect his anonymity from what he perceives as a significant risk of device compromise. There are a number of ways in which an adversary can compromise a device including hacking into a networked device and physically capturing one. For example, if a receiver were to form a sensor network by deploying a set of devices he might be most concerned about an adversary physically capturing and then tampering with a device. Another example is when the receiver initially controls a set of machines that communicate with him over the Internet. In this case the receiver might be concerned about both physical and hacking attacks.

While we find this broad classification helpful, not every scenario distinctly falls into one category. Additionally, while Incomparable Public Keys are useful for protecting the anonymity of the receiver, they do not prevent other types of attacks such as a device feeding the receiver false information.

In both classes of situations we found that Incomparable Public Keys can be used in a variety of ways for communicating with an anonymous receiver. Incomparable Public Keys can be used for secure message delivery, to establish an anonymous secure connection in a key exchange protocol, and in peer-to-peer systems. We describe several ways to apply Incomparable Public Keys later in the paper.

1.4 Structure of this Paper

Section 2 discusses related work, and explains why previously proposed systems fail to provide the desired anonymity properties. Section 3 describes the theory behind Incomparable Public Keys including a definition and an ElGamal-based implementation. We follow in Section 4 by exploring the practical applications of Incomparable Public Keys. We begin by comparing Incomparable Public Keys to several other methods that might hope to achieve anonymity. We analyze the efficiency of using Incomparable Public Keys in a multicast environment and show how they can be adapted for use a key exchange protocol. Next, we describe our prototype implementation of Incomparable Public Keys in the popular GnuPG software package. Finally, we conclude in Section 5.

2. RELATED WORK

Several existing proposals address parts of the receiver anonymity problem, but all have flaws or limitations that prevent them from providing a complete solution.

Pfitzmann and Waidner [14] point out that, to prevent message routing from compromising receiver anonymity, each message should be routed not to a single receiver, but to a multicast address, so the true receiver will retain anonymity among the group of all receivers listening on that address. They further propose marking each message in some way so that the intended receiver can distinguish that message from all other messages sent on the multicast address. Pfitzmann and Waidner refer to this mark as an implicit address if only the intended receiver can distinguish the message as being addressed to him. A mark is called an invisible implicit address if the marks of two messages from the same sender to the same receiver cannot be tested for equality. The result of invisible implicitly marked messages on a multicast address is that no observer should be able to tell the true receiver of a message or be able to link a pair of messages by looking at the mark. Pfitzmann and Waidner observe that a public key cryptosystem can be used to realize invisible implicit addresses, where each possible receiver decrypts an address with their private key and uses message redundancy to determine if the message was destined for him. However, although their proposal provides receiver anonymity against an *eavesdropper*, they do not explore the possibility of withholding information about the receivers' identities from the *senders*.

Chaum [4] and Goldschlag, Reed and Syverson [9, 20] introduced anonymous reply addresses and reply onions. In Chaum's system an initiator may use a chain of nodes to send a message anonymously. The initiator may include an anonymous reply address for the responder to send back a response. The anonymous reply address is a set of layered routing and encryption instructions for sending back a response. Even the responder will not know where he is sending the message by using the address, thus it is anonymous. Chaum's reply addresses (and those of Goldshlag *et al.*) have the drawback that they can only be used once. This means that if a responder has more information to send, but has used up all of the reply addresses, he is unable to send any more messages. (This problem does not occur with sender anonymity since a sender can always generate new layered routing instructions to protect his identity.) Additionally, these systems do not specify how the initiator will identify and decrypt messages if the initiator has a large number of anonymous reply addresses outstanding.

Pseudonym servers, such as the one described by Mazières and Kaashoek [12], supply only pseudo-anonymous addresses. Multiple senders can collude and discover whether they are sending to the same receiver by comparing pseudonym addresses. A receiver might try to get around this by creating and using a unique pseudonym, with a unique public key, for each potential sender. However, that approach would degrade the receiver's efficiency as the receiver would need to have a large number of secret keys, and would have to try decrypting each message with each of its secret keys[2].

Bellare *et al.* formalize a cryptographic security property they name Key-Privacy [1]. Suppose an adversary was given two public encryption keys (pk_0, pk_1) and ciphertext c that was encrypted with one of those two public keys. A cryptosystem maintains Key-Privacy if an adversary cannot determine (with better success than random guessing would provide) which key was used for encryption. We observe that if an asymmetric cryptosystem is used to mark addresses and the Key-Privacy requirement is met, then the addresses will be invisible. Key-Privacy addresses part of the anonymity requirement, by preventing encrypted messages from leaking information about the keys used to encrypt them. However, maintaining Key-Privacy does not necessarily prevent information from leaking to the senders via the keys themselves.

Shields and Levine discuss the use of IP Multicast for receiving anonymous traffic [19]. In their scheme a group of receivers make an anonymous group by all listening to the same IP multicast address. However, they do not discuss the problem of discerning which key to use upon reception of an encrypted message.

Golle *et al.* independently worked on an idea similar to to Incomparable Public Keys, which they call Universal Re-encryption [10]. Universal Re-encryption allows mix-nets to re-encrypt ciphertext without knowing the public key used to encrypt the ciphertext. Using Universal Re-encryption allows the cumbersome key-distribution to be skipped in setting up a mix-net. In their scheme the ciphertext is equivalent to an ElGamal encrypted message with an Incomparable Public Key attached to it. A ciphertext is re-encrypted by re-encrypting the first part with the attached Incomparable Public Key and then transforming the Incomparable Public Key so that it is equivalent to, but computationally indistinguishable from its previous form. Although some of our techniques are similar to those of Golle *et al.*, we focus on how anonymity is affected by the contents of public keys whereas they augment ciphertexts for re-encryption in mix-nets.

Several systems [19, 9, 20, 5, 17, 4, 7] are useful for providing sender anonymity. These systems could be used in a complementary fashion with Incomparable Public Keys. A sender could use the Incomparable Public Key to encrypt the message and then use one of these systems to deliver the encrypted message to a multicast address.

3. INCOMPARABLE PUBLIC KEYS

[2]The sender could attach a tag to each message to specify which key should be used to decrypt that message, but then these tags would leak information to an eavesdropper.

We have discussed how the encryption scheme is an essential element in protecting receiver anonymity. To achieve receiver anonymity we must use an asymmetric key cryptosystem that has the same or similar efficiency as other asymmetric cryptosystems and does not allow multiple senders to be able to determine if they are encrypting messages for the same receiver. In this section we outline the requirements for an Incomparable Public Key scheme.

3.1 Incomparable Public Key Requirements

We specify three requirements that an Incomparable Public Key scheme should have. (Formal definitions appear in Appendix A.)

3.1.1 Generation of Incomparable Public Keys

The holder of a secret key must be able to generate a large number of public encryption keys such that any message encrypted with any of these public keys can be decrypted by one secret key. We call two public keys equivalent if they can be used to encrypt messages for the same secret key. Public keys must have the property that they cannot be tested for equivalence without access to the corresponding secret key(s). A holder of the secret key must be able to generate a practically inexhaustible number of these keys.

When attempting to determine if two keys are equivalent the adversary may try several different attacks including combining the keys and observing the receiver's response to messages. Such a threat model is valid when Incomparable Public Keys are used in applications such as Key-Exchange protocols (see Section 4.2) where the anonymous receiver will respond to messages. To account for these types of attacks we give the adversary access to a decryption oracle in our formal model (Appendix A). This definition constitutes a very strong model of security for Incomparable Public Keys.

For simplicity, we create schemes that are semantically secure, but we claim that it is possible to add security against adaptive chosen ciphertext attacks [15] to our schemes.

3.1.2 Key-Privacy

There must be no way of determining if two encrypted messages were encrypted with the same key. We extend Bellare *et al.*'s [1] definition of Key-Privacy to account for the possibility that two public keys are equivalent. We formally define the Key-Privacy property in Appendix A.2. If the Key-Privacy property holds then a network eavesdropper will be unable to identify two encrypted messages as having been encrypted with the same key.

3.1.3 Efficiency

The efficiency of an Incomparable Public Key scheme must be similar to other asymmetric key schemes. Any scheme in which the work performed to decrypt a message grows with the number of equivalent public keys that might have possibly been used to encrypt the message is not acceptable.

3.2 ElGamal Implementation

We make novel use of the ElGamal cryptosystem to realize an Incomparable Public Key scheme. To create a new Incomparable Public Key a receiver chooses a random generator g and creates the key as (g, g^a) where a is the private key of the receiver. To construct another Incomparable Public Key the receiver chooses another generator h at random and creates the key (h, h^a). These keys are equivalent in function, but a pair of them is indistinguishable from a non-equivalent pair under the Decisional Diffie-Hellman (DDH) assumption. Our scheme's security relies on the number theoretic DDH assumption in the Random Oracle model. We detail our scheme in the rest of this section and offer a formal proof of correctness in Appendix B.

3.2.1 Structure of Keys

All receivers share a prime p where $q = \frac{p-1}{2}$ is also a prime. A newly created public key is an ElGamal public key (g, g^a) where g is a randomly chosen quadratic residue in Z_p^*. The receiver stores the key (g, g^a) in a hash table to record it as being valid. (The records of distributed keys will be used when decrypting a message.) Alternatively, the receiver could keep a private MAC key to itself and give a MAC of the key to the sender that would validate the key. This would alleviate the receiver from the task of recording every key it creates.

3.2.2 Encryption

Suppose a sender holds an Incomparable Public Key (g, g^a). (The key he holds is one of possibly several Incomparable Public Keys corresponding to the private key a.) To encrypt a message the sender will first randomly choose a key K for a symmetric cipher. He will then encrypt the message as $(g^r, g^{ar}K)$,$H(r)$, $E_K(r, (g, g^a), message)$. H is a secure random hash function, r is the random exponent used in the ElGamal part of the encryption, and E_K denotes a random encryption with a symmetric cipher. (The cipher must be semantically secure and should use part of K to authenticate the encryption [2].) This encryption method uses ElGamal with a cryptographic envelope. (K will need to be mapped from the quadratic residues of Z_p^* to a symmetric cipher key.)

3.2.3 Decryption

Suppose a message is received of the form $((d, e), h, M)$. Decryption will proceed as follows.

1. Let $K = \frac{e}{d^a}$. Use K to decrypt the envelope M. This decryption gives us $r, (g, g^a), message$.
2. Check that $h = H(r)$ and that the public key is recorded as a valid one.
3. Check that $g^r = d$ where (g, g^a) is the key in the envelope.
4. If any of these checks fail disregard the message, otherwise the decryption is $message$.

In the encryption process the encryptor uses the hash function to prove knowledge of encryption factors for the public key he was given. Since all valid public keys are recorded, two encryptors cannot combine two public keys to form a valid hybrid key to encrypt with. We note that this attack would be possible in a simpler scheme that did not record valid public keys.

3.2.4 Key-Privacy

Encryption and decryption proceed in exactly the same way as in standard ElGamal with the addition of a cryptographic envelope and a hash function. Since the symmetric encryption key used each time is fresh and the encryption is randomized it will not compromise Key-Privacy. Bellare *et al.* demonstrate that encryption in ElGamal meets this criterion [1].

3.2.5 Efficiency

In the process of decryption, if the message is not rejected before step three the decryption algorithm will require one more exponentiation to finish the validity check. Messages not intended for the receiver will be rejected before this point (unless maliciously created) so the extra cost will only be borne by the intended receiver. Since the exponentiation will dominate the running time, the cost of decryption will be about double that of standard ElGamal when decrypting a message intended for that receiver. When a receiver attempts to decrypt a message intended for someone else, the decryption attempt will be about the same cost as standard ElGamal.

The cost for encryption will be approximately the same as in standard ElGamal.

3.3 An Implementation Using General Assumptions

The implementation in the previous section is an efficient construction of an Incomparable Public Key scheme. The proof is heuristic since it is in the Random Oracle model. We note that a similar scheme can be derived from general assumptions, with no need for the Random Oracle model, but with a minor added cost in running time. This is a two-key scheme based on the two-key paradigm of Naor and Young [13]. In this method the Incomparable Public Key will consist of two randomly generated ElGamal Keys $(g_1, g_1^a), (g_2, g_2^b)$. An encrypted message will be of the form $(g_1^r, g_1^{ar}K)$, $(g_2^r, g_2^{br}K)$, $E_K(r, (g_1, g_1^a), (g_2, g_2^b), message)$. We give a detailed description of this construction and a proof of security in Appendix C.

3.4 Light Version for Passive Receivers

We created our definition with a threat model that included receivers replying to senders. In some circumstances a receiver will receive messages and not give any indication to a sender whether messages were received correctly or not. In this case we can relax our definition to not include a decryption oracle. We call receivers that behave like this Passive Receivers.

A lighter version of our protocol exists for Passive Receivers. The key is again a randomly generated ElGamal key (g, g^a), but the ciphertext is just $(g, g^{ar}K), E_K(message)$. The proof of incomparability is derived directly from the Decisional Diffie-Hellman assumption.

4. INCOMPARABLE PUBLIC KEYS IN PRACTICE

4.1 Comparison with Other Methods

In this section we consider some alternative cryptographic methods that might hope to achieve anonymity in the same way as Incomparable Public Keys. We examine the significant differences between each of these and an Incomparable Public Key scheme and find that each of the alternative methods considered in this section is deficient in some respect compared to our method.

For each technique we will assume that the message is sent to a multicast address.

4.1.1 Standard Asymmetric Key Scheme

One encryption alternative to using Incomparable Public Keys is to use a standard asymmetric key scheme for encryption. The receiver could generate one asymmetric key pair and distribute the public key to all senders. However, this allows the senders to determine that they are sending to the same receiver, in violation of the requirements. If the receiver leaks some information about its identity to each sender, this method would allow the senders to aggregate that information and learn too much about the receiver's identity.

4.1.2 Several Independent Symmetric Keys

Another alternative is to give each sender a unique symmetric encryption key, and have the receiver try decrypting each message with each of these symmetric keys. In this approach, decryption time is linear in the number of senders; however if there are not too many senders the performance might still be better than an Incomparable Public Key scheme, which uses asymmetric key operations. Another problem is that if an adversary gained access to the sender's key he would be able to read all messages sent by that sender in the past.

4.1.3 Several Independent Public Keys

We could take a similar approach to the one above by having the receiver generate a fresh asymmetric key-pair for every sender. The receiver would give a new public key to every sender and keep the new secret key. Then for every message sent to the multicast address the receiver would need to try all possible decryption keys until one worked or all were exhausted. If an adversary gained access to the sender's key he would still be unable to read past messages sent by the sender. In all other respects this method has all of the drawbacks of the previous one.

4.1.4 Message Markers

Another possible method would be for the receiver to give the senders a marker for identifying their messages and a unique (symmetric or asymmetric) encryption key to decrypt messages. These markers could be used by the receiver to determine which decryption key to use. This technique has the advantage that the receiver need not waste time decrypting messages that are not intended for him and that at most one decryption is needed.

The same marker cannot be used twice without alerting an eavesdropper that the two messages have the same destination, so a fresh marker would need to be generated and used for each message. This requirement of a fresh marker creates a need for synchronization between sender and receiver, which would apparently require additional message traffic between them that could endanger anonymity. For example, if the markers are random nonces, then communication would have to be two-way, so that the receiver could periodically send along new nonces.

4.1.5 Summary of Comparisons

Through these comparisons we can see the advantages of using an Incomparable Public Key scheme to realize an anonymous identity. First, an Incomparable Public Key scheme allows true receiver anonymity as opposed to pseudo-anonymity. Second, the work spent per message does not increase with the number of potential senders of the message.

An anonymous identity that uses Incomparable Public Keys can be used repeatedly to encrypt and send messages without communication from the receiver. Finally, using an Incomparable Public Key scheme is robust in the face of message loss. If any number of messages are lost, the receiver will still be able to read the next one that gets through.

4.2 Efficiency in Practice

We analyze the efficiency of receiving messages in a system that uses Incomparable Public Keys as follows. Suppose there are r receivers on a channel and (for simplicity) n messages are sent to each particular receiver per second. Then each receiver will need to perform rn decrypt operations per second. (ElGamal (1024 bit) decryption in a particular software package was measured to take $\sim$ 5.8ms on an 850Mhz Celeron processor [6]. This implies that a conventional machine can decrypt about 170 messages per second.) One can see how there is a simple trade-off of anonymity, measured by r, and efficiency, measured by n. Suppose an Incomparable Public Key scheme were not used and an independent public/private key pair were generated for every possible sender of a message. If there were m senders per receiver then the efficiency would go down by approximately a factor of m as all m private keys would have to be tried for each incoming message.[3]

This analysis applies to a simple multicast channel. Incomparable Public Keys can also be used in other environments. For example, the protocol $\mathcal{P}^5$ is a peer-to-peer system that allows participants to tradeoff efficiency in communication for increased anonymity by placing themselves in certain communication groups [18]. The efficiency analysis in these systems is more complicated, but if Incomparable Public Keys are used, then each receiver only needs to execute one decryption operation per incoming message.

4.2.1 Secure Sessions via SKEME Key Exchange

In practice, parties will often initiate a key exchange using public-key cryptography to derive a temporary shared symmetric key. The parties can then communicate over a secure session using the shared symmetric key. The communication over the session will then be efficient since symmetric key cryptography is used.

Incomparable Public Keys can be used to initiate secure key exchanges. We illustrate this by describing how the popular SKEME [11] key exchange protocol can be modified to include Incomparable Public Keys. The SKEME protocol is diagrammed in Figure 1. In the diagrammed version the initiator, I, engages in a Diffie-Hellman key exchange. The initiator verifies the Diffie-Hellman component sent from the responder is authentic using public key encryption.

We can use Incomparable Public Keys to protect responder anonymity. In the first step the initiator will send the initiation message to a multicast address and encrypt it with an Incomparable Public Key. Of all the listeners on the multicast channel only the intended responder will be able to correctly decrypt the message.

After the completion of these steps the parties will share a session key. In continuing the session the initiator will send messages to the multicast channel. However, the responder still has the problem of determining which session messages that were encrypted with symmetric key messages belong to him. One of the alternative methods listed in Section 4.1 might be used for this.

Even though we argued against such methods for establishing communication, they could be useful for maintaining an open session. We derive this from the fact that the number of potential parties that may contact a host will typically be much larger than the number of sessions open at any one time. Thus we see that Incomparable Public Keys can be used for the initiation of a session when the number of possible initiators is large, after which we can switch to method that is more efficient for open sessions.

$$
\begin{array}{lcr}
I & & R \\
& \xrightarrow{\quad E_{K_R}(K_A, K_I),\, s^x \quad} & \\
& & K_0 = H(K_A, K_B) \\
& \xleftarrow{\quad E_{K_I}(K_B),\, s^y,\, HMAC_{K_0}(s^x, s^y) \quad} & \\
K_S = H(s^{xy}) & &
\end{array}
$$

Figure 1: The SKEME protocol where only the responder is authenticated. The parties use K_S as their session key that was derived from a Diffie-Hellman key exchange. The exchange is authenticated using public key encryption instead of signing.

4.3 Implementation

In this section, we describe a prototype implementation that demonstrates how our method could be used in practice. We chose to implement the prototype by extending an existing application in order to show that our method is easily adapted to the needs of established applications.

We implemented our scheme by modifying the popular Pretty Good Privacy (PGP) encryption software suite [3]. In particular, we modified GnuPG 1.2.0 [21], a free software replacement for PGP. PGP already enjoys a large number of users, many of whom are interested in protecting receiver anonymity. Some of these users direct senders to send encrypted messages to anonymous newsgroups such as *alt.anonymous.messages*. The receiver attempts to maintain anonymity among all of the readers of the newsgroup. These newsgroups can be viewed as multicast addresses.

Our prototype uses the ElGamal-based encryption scheme of Section 3.4. Fortunately, the GnuPG software already included an implementation of the standard ElGamal algorithm, so we were able to reuse most of the existing code. Our Incomparable Public Key scheme is presented to the user as a separate algorithm, leaving the original signing and encryption algorithms intact.

The most significant modifications we made to the code were in the public key export function. PGP's public key export function allows a user to take a public key from his public key database and export it into a text file. Another user may then add that public key to their database by "importing" the public key from that file into their database. We modified the key export function so that it wrote into the file a newly generated public key that is incomparable but equivalent to the one in the user's database. The ef-

[3] This is not exactly true because after a successful decryption of a message the receiver would not try any more of its keys. However, as r becomes large this approximation becomes correct.

fect of this is that whenever a user passes a public key to another user, the key that is passed will be a new, incomparable one; so different users' databases will contain different, Incomparable Public Keys for the same receiver.

A few other code changes were required. For example, we changed the code to use the same (suitably chosen) 1024-bit prime as the modulus in all Incomparable Public Keys, so as to avoid leaking information in the modulus. PGP also attaches to each key an identifying user ID, consisting of the key owner's name and e-mail address. We effectively removed the user ID by modifying the code to attach the same dummy user ID to all keys. A sender who has multiple keys to manage can still refer to them by their key fingerprints.

The encryption and decryption functions were kept unchanged. PGP by default appends the fingerprint of the key used for encryption to an encrypted message. A user can invoke an option to disable this feature and encrypt in what PGP calls "anonymous receiver" mode. We enforce the use of this option for all messages encrypted with an Incomparable Public Key.

The code for our implementation is available at `http://www.cs.princeton.edu/~bwaters/research`.

5. SUMMARY

We have seen that the correct choice of an encryption scheme is a crucial aspect to providing true receiver anonymity. Current encryption techniques either allow keys to be compared, which degrades anonymity, or require the receiver to try a large number of keys for each decryption. We solve this problem by introducing a new type of encryption scheme, which we call an Incomparable Public Key scheme. Using an Incomparable Public Key scheme a receiver can generate a large number of equivalent public keys. Anonymity is maintained since public keys cannot be tested for equivalence. The scheme is efficient because only one secret key needs to be used for decryption. We were able to realize an Incomparable Public Key scheme by making novel use of the ElGamal cryptosystem.

We then analyzed our scheme in a practical setting and found that it offered the best combination of efficiency and true receiver anonymity for when there were many possible senders. Finally, we implemented our Incomparable Public Key scheme into the popular GnuPG software encryption suite.

6. ACKNOWLEDGMENTS

We would like to thank Lujo Bauer, Dan Boneh, Carl Kingsford, Ruoming Pang, and Iannis Tourlakis for helpful suggestions over the course of this project. We are also grateful for the advice of the anonymous reviewers.

7. REFERENCES

[1] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and D. Pointcheval. Key-privacy in public-key encryption. *Lecture Notes in Computer Science*, 2248, 2001.

[2] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm. *Advances in Cryptology - Asiacrypt 2000 Proceedings, Lecture Notes in Computer Science*, 1976, 2000.

[3] Jon Callas, Lutz Donnerhacke, Hal Finney, and Rodney Thayer. RFC 2440: OpenPGP message format, November 1998. Status: PROPOSED STANDARD.

[4] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Proceedings of Communications of the ACM*, 24(2):245–253, 1981.

[5] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology 1(1)*, pages 65–75, 1988.

[6] Wei Dai. Crypto++ 4.0 benchmarks. `http://www.eskimo.com/~weidai/benchmarks.html`.

[7] George Danezis, Roger Dingledine, David Hopwood, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol, 2002. `http://mixminion.net`.

[8] Tahir ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *Advances in Cryptology Proceedings of CRYPTO 84*, pages 10–18, 1985.

[9] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM (USA)*, 42(2):39–41, 1999.

[10] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. Universal Re-encryption for Mixnets, 2003. `http://crypto.stanford.edu/~pgolle/papers/univrenc.html`.

[11] Hugo Krawczyk. SKEME: A versatile secure key exchange mechanism for the Internet. In *Symposium on Network and Distributed Systems Security*, pages 114–127, 1996.

[12] David Mazières and M. Frans Kaashoek. The design, implementation and operation of an email pseudonym server. *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 27–36, 1998.

[13] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the 22nd Annual Symposium on Theory of Computing*, 1990.

[14] Andreas Pfitzmann and Michael Waidner. Networks without user observability. *Lecture Notes in Computer Science*, 219:245–253, 1986.

[15] Charles Rackoff and Daniel Simon. Non-interactive zeroknowledge proof of knowledge and chosen ciphertext attack. *Advances in Cryptlogy CRYPTO '91, Lecture Notes in Computer Science*, 576, 1991.

[16] Josyula R. Rao and Pankaj Rohatgi. Can pseudonymity really guarantee privacy? In *Proceedings of the Ninth USENIX Security Symposium*, pages 85–96. USENIX, August 2000.

[17] Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[18] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A protocol for scalable anonymous communication. In *IEEE Symposium on Security and Privacy*, 2002.

[19] Clay Shields and Brian Levine. A protocol for anonymous communication over the internet. In *Proceedings of the 7th ACM Conference on Computer and Communication Security*, Athens, Greece, 2000.

[20] Paul Syverson, David Goldschlag, and Michael Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, California, 4–7 1997.

[21] The GNU Privacy Guard. `http://www.gnupg.org`.

APPENDIX

A. INCOMPARABLE PUBLIC KEY DEFINITION

In this section we formally define the Generation of Public Keys requirement and the Key-Privacy requirement given in Section 3.

For the discussion we will use the following notation. G will serve as a common key generator that given a security parameter k will produce the common key I. I will serve as a common global parameter for key generation. K is a secret key generation algorithm that given I will randomly generate a secret key. L is an algorithm that given a private key will randomly generate a corresponding Incomparable Public Key.

A.1 Generation of Public Keys

We define the following two experiments. We derive $I \overset{R}{\leftarrow} G(k)$ as the global parameter shared by both experiments.

$$\begin{array}{l} \text{Experiment1}(I, K, L) \\ \quad sk_0 \overset{R}{\leftarrow} K(I) \\ \quad pk_0 \overset{R}{\leftarrow} L(sk_0) \\ \quad pk_1 \overset{R}{\leftarrow} L(sk_0) \\ \quad \mathbf{output}(\mathbf{pk_0}, \mathbf{pk_1}) \end{array}$$

$$\begin{array}{l} \text{Experiment2}(I, K, L) \\ \quad sk_0 \overset{R}{\leftarrow} K(I) \\ \quad sk_1 \overset{R}{\leftarrow} K(I) \\ \quad pk_0 \overset{R}{\leftarrow} L(sk_0) \\ \quad pk_1 \overset{R}{\leftarrow} L(sk_1) \\ \quad \mathbf{output}(\mathbf{pk_0}, \mathbf{pk_1}) \end{array}$$

The Generation of Public Keys requirement is then satisfied if no (computationally bounded in poly-time(k)) adversary A can gain more than a negligible advantage when attempting to distinguish the output from the two experiments. We give the adversary a decryption oracle, $\mathcal{O}$ and denote the adversary as $A^{\mathcal{O}}$.

In Experiment1 the two equivalent public keys were derived from the same private key, whereas in Experiment2 two non-equivalent public keys were generated from two different private keys that were generated to the same set of global parameters.

A.2 Key-Privacy

We require that an Incomparable Public Key scheme meets Bellare *et al.*'s Key-Privacy requirement [1]. Specifically, we require the indistinguishability of keys under chosen plaintext attack. Suppose an adversary is given two public keys and a ciphertext message that was encrypted with one of the public keys. The Key-Privacy requirement states that an adversary will have at most a negligible advantage in determining which key was used for encrypting the message.

In Bellare *et al.*'s definition the authors define an experiment where the two public keys are both generated randomly and independently. We also want the Key-Privacy property to hold for the case when two public keys are chosen randomly, but derived from the same private key (i.e., the keys are equivalent). We define two experiments to define Key-Privacy as it pertains to Incomparable Public Keys. The first one is the same as Bellare *et al.*'s and the second one is adjusted slightly to account for equivalent public keys.

Let A be any (computationally bounded in poly-time(k)) adversary, x be a message chosen by the adversary to be encrypted, s be state information that the adversary uses, and E be the encryption algorithm. $I \overset{R}{\leftarrow} G(k)$ is the common global parameter.

$$\begin{array}{l} Experiment_{non-equivkeys}(b, I, K, L) \\ \quad sk_0 \overset{R}{\leftarrow} K(I) \\ \quad sk_1 \overset{R}{\leftarrow} K(I) \\ \quad pk_0 \overset{R}{\leftarrow} L(sk_0) \\ \quad pk_1 \overset{R}{\leftarrow} L(sk_1) \\ \quad (x, s) \overset{R}{\leftarrow} A(\text{“find x”}, pk_0, pk_1) \\ \quad y \overset{R}{\leftarrow} E_{pk_b}(x) \\ \quad d \overset{R}{\leftarrow} A(\text{“guess b”}, y, s) \\ \quad \mathbf{output}(\mathbf{d}) \end{array}$$

A scheme is secure if all adversaries have at most a negligible advantage in guessing b.

The second definition is as follows.

$$\begin{array}{l} Experiment_{equivkeys}(b, I, K, L) \\ \quad sk_0 \overset{R}{\leftarrow} K(I) \\ \quad pk_0 \overset{R}{\leftarrow} L(sk_0) \\ \quad pk_1 \overset{R}{\leftarrow} L(sk_0) \\ \quad (x, s) \overset{R}{\leftarrow} A(\text{“find x”}, pk_0, pk_1) \\ \quad y \overset{R}{\leftarrow} E_{pk_b}(x) \\ \quad d \overset{R}{\leftarrow} A(\text{“guess b”}, y, s) \\ \quad \mathbf{output}(\mathbf{d}) \end{array}$$

Both public keys are derived from the same private key. Again a scheme is secure if all adversaries have at most a negligible advantage in guessing b.

Bellare *et al.* show that the first definition of Key-Privacy holds for the ElGamal cryptosystem if the cryptosystem is secure. The second definition of Key-Privacy will hold unconditionally for two equivalent ElGamal keys since any particular encryption of a message is equally likely to have come from either one of two equivalent keys.

B. PROOF FOR ELGAMAL IMPLEMENTATION

We re-examine the experiment defined in Appendix A.1 with the ElGamal implementation substituted in. We assume the global parameter $I = p$ (the prime component of a key) has already been generated.

$$\begin{array}{l} \text{Experiment1}(I, K, L) \\ \quad a = sk_0 \overset{R}{\leftarrow} K(I) \\ \quad (g, g^a) = pk_0 \overset{R}{\leftarrow} L(sk_0) \\ \quad (g', g'^a) = pk_1 \overset{R}{\leftarrow} L(sk_0) \\ \quad \mathbf{output}(\mathbf{pk_0}, \mathbf{pk_1}) \end{array}$$

$$\begin{aligned}&\text{Experiment2}(I, K, L)\\&\quad a = sk_0 \xleftarrow{R} K(I)\\&\quad a' = sk_1 \xleftarrow{R} K(I)\\&\quad (g, g^a) = pk_0 \xleftarrow{R} L(sk_0)\\&\quad (g', g'^{a'}) = pk_1 \xleftarrow{R} L(sk_1)\\&\quad \mathbf{output}(\mathbf{pk_0}, \mathbf{pk_1})\end{aligned}$$

In each distribution a decryption oracle, $\mathcal{O}$, is provided. The oracle is instantiated with private keys and with one or more valid public keys for each private key. The oracle, when give a ciphertext, will output a list of valid decryptions of that ciphertext given for the keys it has.

We prove that there cannot exist a (poly-time) distinguisher, $A^{\mathcal{O}}$, between the two experiments. We prove this by contradiction. Suppose there is a distinguisher between the two experiments. We can take advantage of it in the following manner. We will build a distinguisher that can tell if a quadruple is a Diffie-Hellman quadruple[4] with greater than negligible advantage. Our technique is to take the quadruple (w_1, w_2, w_3, w_4) and call the distinguisher with $A^{\mathcal{O}}((w_1, w_2), (w_3, w_4))$. To use the distinguisher we will also need to be able to simulate the decryption oracle in the Random Oracle model (we show how to do this in the next subsection). We notice that if $A^{\mathcal{O}}$ can distinguish between the two experiments this corresponds exactly with our distinguisher being able to distinguish between Diffie-Hellman quadruples, which is assumed to be hard. We now show that we can simulate the decryption oracle to complete our proof.

B.1 Simulator Construction

We will build a simulator of the decryption oracle described above. The simulator is initiated with the public keys (w_1, w_2) and (w_3, w_4).

The decryption oracle simulator acts as follows. When a query is made to the random oracle, H, the simulator randomly chooses a bitstring, h, as a response and records the query/response pair in a table. The chances of a collision are negligible in the security parameter.

A decryption query is of the form $((d, e), h, M)$. The simulator will have access to the two public keys and the oracle query table, but not the corresponding private key(s). For both public keys the simulator will take the following actions. For convenience we describe the actions in terms of the first key $(w_1, w_2) = (g, g^a)$ for some a. Keep in mind that the simulator does not know a. The simulation will be repeated for the key (w_3, w_4).

1. Look up h in the response half of the random oracle query table and set r to be the corresponding query. If no entry for h exists, reject the decryption. The chances of success are negligible if the oracle was not consulted to get h.
2. Compute $K = \frac{e}{(g^a)^r}$ and decrypt the envelope, M, to get r', PublicKey, message.
3. Reject if $r' \neq r$ or if the public key from the envelope is not the one being used in the simulation.
4. Reject if $d \neq g^r$.
5. If not rejected, output *message*.

[4]A quadruple is a Diffie Hellman quadruple if it is of the form (g_1, g_1^a, g_2, g_2^a) where g_1 and g_2 are generators of the group.

We now show that our simulator is equivalent to the true decryption oracle with very high probability. The true decryption oracle will have access to the valid public keys and the private keys. Our simulator has access to the valid public keys and runs the random oracle. The simulator and decryption oracle are equivalent if every message that is output by the true oracle is output by the simulator and every message output by the simulator is output by the true oracle.

Suppose the true decryption oracle accepts and outputs a certain message. The cryptographic envelope would then have to contain a valid public key (g, g^a) that our simulator would have access to. The envelope would also contain r where $h = H(r)$. Our simulator would (with very high probability) have derived the same r from h and the random oracle query table. The final check of the true decryption oracle is that $d = g^r$. The true oracle gets the decryption envelope key K by dividing e by $d^a = (g^r)^a = (g^a)^r$. However, in our simulation for key g, g^a we will have divided e by $(g^a)^r$ and then performed the same authenticity checks. Therefore any messages that are decrypted by the true oracle are decrypted by our simulator with very high probability.

Suppose a message were decrypted by our simulator using the hash table and public key (g, g^a). Then the simulator is able to get r from h and the random oracle table. The simulator gets the envelope key K by dividing e by $(g^a)^r$. However, after the simulator decrypts the envelope it checks that the public key being used is g, g^a and that $d = g^r$. The true decryption oracle that has the secret key a will divide e by $d^a = g^{ar}$ and compute the same K as the simulator. The simulator is able to perform the same authenticity checks as the decryption oracle. The simulator simulates the decryption oracle correctly since with high probability the simulator will decrypt a message if and only if the true decryption oracle will.

C. TWO-KEY IMPLEMENTATION

In Section 3.3 we stated that there existed a construction of an Incomparable Public Key scheme for which the security proof could be made outside the Random Oracle model. We now present a detailed description of the construction and a proof of the Incomparable property.

C.1 Description

C.1.1 Structure of Keys

All receivers share a prime p where $q = \frac{p-1}{2}$ is a prime. The public key consists of two ElGamal public keys (g_1, g_1^a), (g_2, g_2^b) where g_1 and g_2 are quadratic residues in Z_p^*. Again the receiver records the public key pair to mark them as being valid.

C.1.2 Encryption

To encrypt a message the sender will first randomly choose a key K for a symmetric cipher. He will then encrypt the message as $(g_1^r, g_1^{ar}K)$, $(g_2^r, g_2^{br}K)$, $E_K(r, (g_1, g_1^a), (g_2, g_2^b),$ *message*$)$.

C.1.3 Decryption

Suppose a message is received of the form $((d, e), (i, j), M)$. Decryption will proceed as follows.

1. Let $K = \frac{e}{d^a}$. Also check that the same $K = \frac{j}{i^a}$ Use K to decrypt the envelope M. This decryption gives us $r, g_1, g_1^a, g_2, g_2^b, message$.
2. Check that the public keys inside the envelope have been recorded as a valid ones.
3. Check that $g_1^r = d$, $g_2^r = i$ where (g_1, g_1^a, g_2, g_2^b) is the key in the envelope.
4. If any of these checks fail, disregard the message; otherwise the decryption is $message$.

C.1.4 Key-Privacy

Key-Privacy follows for the same reason as in original construction.

C.1.5 Efficiency

Encryption requires two exponentiations to use both components of the public key. A successful decryption requires four exponentiations, however, an unsuccessful decryption can be detected after one exponentiation.

C.2 Proof for two-key implementation

Again the common global parameter $I = p$ where p is a strong prime. We define three experiments.

Experiment1(I, K, L)
$a, b = sk_0 \xleftarrow{R} K(I)$
$(g_1, g_1^a, g_2, g_2^b) = pk_0 \xleftarrow{R} L(sk_0)$
$(g_1', {g_1'}^a, g_2', {g_2'}^b) = pk_1 \xleftarrow{R} L(sk_0)$
$\mathbf{output(pk_0, pk_1,)}$

Experiment2(I, K, L)
$a, b = sk_0 \xleftarrow{R} K(I)$
$a', b' \xleftarrow{R} K(I)$
$a', b = sk_1$
$(g_1, g_1^a, g_2, g_2^b) = pk_0 \xleftarrow{R} L(sk_0)$
$(g_1', {g_1'}^{a'}, g_2', {g_2'}^b) = pk_1 \xleftarrow{R} L(sk_1)$
$\mathbf{output(pk_0, pk_1)}$

Experiment3(I, K, L)
$a, b = sk_0 \xleftarrow{R} K(I)$
$a', b' = sk_1 \xleftarrow{R} K(I)$
$(g_1, g_1^a, g_2, g_2^b) = pk_0 \xleftarrow{R} L(sk_0)$
$(g_1', {g_1'}^{a'}, g_2', {g_2'}^{b'}) = pk_1 \xleftarrow{R} L(sk_1)$
$\mathbf{output(pk_0, pk_1)}$

For each experiment the adversary, $A^{\mathcal{O}}$, will be given an oracle that knows the private and public keys from the experiment. Again it will output a list of all legal decryptions of a ciphertext when given that ciphertext as input.

Suppose that an adversary can break our scheme and thus distinguish between Experiment1 and Experiment3. Then the same adversary can either distinguish between Experiment1 and Experiment2 or between Experiment2 and Experiment3 (or both). We prove this is impossible by contradiction. First, let's suppose we are in the case where $A^{\mathcal{O}}$ can distinguish between experiments one and two. We will build a distinguisher than can tell if a quadruple is a Diffie-Hellman quadruple. Let our quadruple be (w_1, w_2, w_3, w_4). We will then call $A^{\mathcal{O}}((w_1, w_2, g_2, g_2^b), (w_3, w_4, g_2', {g_2'}^b))$, where g_2,g_2' and b are chosen randomly. If the quadruple is a Diffie-Hellman quadruple then we passed the adversary input from Experiment1, otherwise we passed an input drawn from Experiment2. Thus, if we can simulate the decryption oracle we can distinguish Diffie-Hellman quadruples, which is assumed to be hard.

C.3 Simulator Construction

Using similar arguments to those of Appendix B.1 we build a simulator of the decryption oracle described above. The simulator is initiated with input from one of the experiments. It knows the public keys (w_1, w_2, g_2, g_2^b) and $(w_3, w_4, g_2', {g_2'}^b)$ and the private key b.

The simulator will be able to use the knowledge of the private key b to begin decryption of the envelope, and will use the information inside to check the validity of the message.

A decryption query is of the form $((d, e), (i, j), M)$. For both public keys the simulator will take the following actions. For convenience we describe the actions in terms of the first key $(w_1, w_2, g_2, g_2^b) = (g_1, g_1^a, g_2, g_2^b)$ for some a that is unknown to the simulator.

1. Compute $K = \frac{j}{i^b}$ and decrypt the envelope to get r', PublicKey, message.
2. Reject if the public key inside the envelope is not the one being used in the simulation.
3. Reject if $d \neq g_1^{r'}$,$i \neq g_2^{r'}$, or $K \neq \frac{e}{(g_1^a)^r}$.
4. If not rejected output $message$.

We now show that the output from the simulator matches the output from a true decryption oracle. Suppose the true decryption oracle accepted a message. Then the key K that was used by the true decryption oracle to decrypt the envelope could be derived from either of the two ElGamal encryptions. Otherwise the message would have been rejected by the decryption oracle.

The simulator can use its knowledge of the private key, b, to derive K from the second ElGamal encryption. Once it opens the envelope using K it will perform the same validity checks that the true random oracle performs. Additionally, it will use the knowledge of r and the public keys to check that the both ElGamal encryptions are encryptions of the same K. This ensures that the simulator doesn't accept any messages that the true oracle rejects. Likewise, the oracle only accepts messages that the simulator does. Therefore, the simulator is equivalent to the decryption oracle.

C.4 Case 2

There is also the possibility that the adversary can only distinguish between Experiment2 and Experiment3. In that case we call $A^{\mathcal{O}}((g_1, g_1^a, w_1, w_2), (g_1', {g_1'}^{a'}, w_3, w_4))$ when given a possible Diffie-Hellman quadruple (w_1, w_2, w_3, w_4). If it is a Diffie-Hellman tuple, the input matches Experiment2, otherwise it matches Experiment3. The decryption simulator will know the public keys that were input to the adversary and the private keys a, a'. The simulator is built analogously to the one described above by using the private keys it knows as hints for decryption.