Number Theoretic Attacks On Secure Password Schemes

Sarvar Patel
Bellcore
Math and Cryptography Research Group
445 South St, Morristown, NJ 07960, USA
sarvar@bellcore.com

Abstract

Encrypted Key Exchange (EKE) [1, 2] allows two parties sharing a password to exchange authenticated information over an insecure network by using a combination of public and secret key cryptography. EKE promises security against active attacks and dictionary attacks. Other secure protocols have been proposed based on the use of randomized confounders [4, 7].

We use some basic results from number theory to present password guessing attacks on all versions of EKE discussed in the paper [1] and we also offer countermeasures to the attacks. However, for the RSA version of EKE, we show that simple modifications are not enough to rescue the protocol. Attacks are also presented on half encrypted versions of EKE. We also show how randomized confounders cannot protect Direct Authentication Protocol and Secret Public Key Protocol versions of a secure password scheme [4] from attacks. We discuss why these attacks are possible against seemingly secure protocols and what is necessary to make secure protocols.

1 Introduction

Protocol design has advanced to the stage where we can securely authenticate and agree on session keys over an insecure network. Both passive and active attacks can be thwarted, provided of course that random keys are chosen. Unfortunately, humans choose bad passwords [6], hence all protocols which rely on keys derived from user chosen passwords are susceptible to off-line dictionary attacks. For example, after listening to a challenge R and the password encrypted response $P(R)$, the eavesdropper, off-line, encrypts R with all likely passwords P' from the dictionary and compares the result $P'(R)$ with $P(R)$. If one of the passwords in the dictionary matches, then the secret password has been discovered. One can try to hide the relationship of R and $P(R)$ by encrypting not R but a complex function $f(R)$, or encrypt R not just by P but some function $f(P)$. All of these attempts are still susceptible to password cracking by off-line dictionary attacks because f is known and each password guess from the dictionary can be verified or rejected. Section 2 briefly describes EKE, while section 3, 4, and 5 look at specific variants of EKE and attacks on them. In section 6 we look at another secure password scheme [4] and present attacks on two versions of the protocol. In section 7 we look back and consider why our attacks have been successful against protocols which appear by all accounts to be secure.

2 Encrypted Key Exchange (EKE)

Encrypted Key Exchange (EKE) is different from other password-based protocols because it promises security not only against active attacks but also against off-line dictionary attacks. Bellovin and Merritt's [1] interesting and novel protocols use a combination of secret-key encryption and public key encryption for authentication and key agreement over an insecure network. The basic idea of EKE is to guarantee that trial passwords cannot be verified by information exchanged over the network. This is accomplished by encrypting a randomly generated public key, so that its decryption by trial passwords will result in a random number useless in verifying the trial password. The random session key to be exchanged can now be encrypted by the public key which in turn is encrypted by the shared password. Breaking the public key is considered computationally prohibitive. The ideas will become clearer as we look at specific variants of EKE. The original paper [1] presented three variants: RSA EKE, Diffie-Hellman EKE and ElGamal EKE. For each variant of EKE, we briefly describe the protocol and present attacks on them.

3 RSA Version

3.1 Brief Description of RSA

To encrypt a message M into a ciphertext C, the RSA [9] public key system raises M to the e th power modulo n, $C \equiv M^e \bmod n$. For decrypting, the RSA system raises the ciphertext C to another power d modulo n, $M \equiv C^d \bmod n$.

The numbers e and n form the public key whereas d is the private key. The number n is composed of two large prime factors p and q, while e is chosen to be relatively prime to the Euler phi function $\varphi(n) = (p-1)(q-1)$. Factoring numbers with large primes is a hard problem and no faster way is known to break RSA other than factoring n into its prime components p and q. The decryption exponent d is chosen so that $ed \equiv 1 \pmod{\varphi(n)}$.

3.2 Description of RSA-EKE

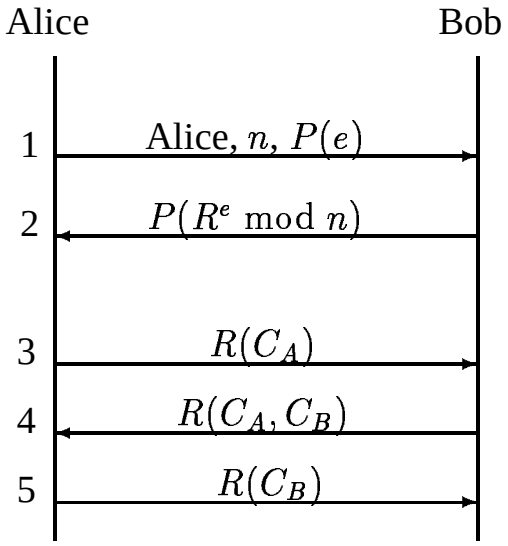

Figure 1. EKE using RSA

In step 1 of the RSA scheme (Figure 1), Alice generates a random public key and the corresponding private key d. From the above description we know that RSA's public key consists of a pair of numbers e and n. Alice sends Bob her name, the number n, and $P(e)$, the public exponent e encrypted by a secret key system using the common shared password P as the secret key. We will explain in section 3.5 why n is sent in the clear. Bob knowing the password P can decrypt $P(e)$ and retrieve e. In step two, he generates a random session key R and encrypts it with Alice's public key to compute $R^e \bmod n$. This is further encrypted using the shared password P and the final result $P(R^e \bmod n)$ is sent to Alice.

Alice can now retrieve R by first decrypting the symmetric encryption $P^{-1}(P(R^e)) = R^e$, and then using her private key to perform $((R^e)^d \bmod n)$ to retrieve R. Now both Alice and Bob know the session key R, and can proceed with standard challenge/response steps to authenticate the session key R. However, we do not describe these further steps necessary to protect against replay attacks.

We assume for discussion purposes that the real password matches one of the W passwords in the attacker's dictionary; although this is not always true, an alarmingly high fraction of the actual passwords match passwords in a constructed dictionary [6]. How does EKE protect us from dictionary attacks? The only way to verify a password candidate P' is to either break the public key system or break the secret key system assuming that the size of n and R are sufficiently large to prohibit exhaustive searching.

Let us assume that the attacker is in possession of some known plaintext X and its encryption under session key R, $R(X)$. If we can go from the W possible passwords P' to W possible session keys R' then we can verify each one to see if $R'(X)$ matches $R(X)$. Fortunately, this will not be possible. Though we can compute the W possible public keys, e, we cannot narrow the space of R down to W numbers. Possessing $P(e)$ and $P(R^e)$, we can try each of W passwords P' and decrypt to e' and $(R^e)'$. Unless we can break the RSA system and find d, this gives us no extra information about possible Rs.

3.3 Subtleties of RSA-EKE

The public exponent, e, of RSA is always an odd number which could leak information. An attacker can try to decrypt $P(e)$ by candidate passwords P'. If the result $P'^{-1}(P(e))$ is an even number then we can immediately reject all such passwords. On average, half of the passwords in the dictionary can be so eliminated. After overhearing more valid sessions, the possible passwords can, at a logarithmic rate, be narrowed down to one. The solution [1] to this problem is to randomly add one to e before encrypting it with P. The user at the other end can remove the one if it was added because the user knows e should always be odd. The attacker on the other hand cannot rule out any passwords P' because an even number may still be a valid e if a one was added to it.

Information leakage can also result from fitting numbers of maximum size n into a block of size 2^m because a trial P' generating a decryption greater than n can be rejected. We will not base our attacks in the paper on this leakage because it is already discussed in another place [8].

3.4 Classification of Attackers

Depending upon the power of the attacker, different kinds of attacks are possible. We classify attackers according to their abilities into three types. First, the least powerful attacker is the **querying attacker**. The only capability this attacker possesses is to initiate sessions with Bob while pretending to be Alice. The reactions and responses of Bob are used by the attacker to discover the password. Neither spoofing of network addresses, nor eavesdropping on other sessions is assumed. Only the requisite functionalities to initiate the protocol and use the response are needed. Next in power is the **eavesdropping attacker**. This attacker has the power to eavesdrop on other legitimate sessions by perhaps placing himself in the path of the two parties. And finally we have the **active attacker** who has the ability to intercept packets and insert his own packets. As an example, the active attacker can assume the identity of Bob, stop any packets Alice sends Bob, and while pretending to be Bob carry a session with Alice. Practically all the attacks we describe only require the abilities of a querying attacker.

3.5 Number Theoretic Attack On RSA-EKE

We are now in a position to understand why n could not have been sent encrypted because any password resulting in a decryption with small prime factors could be rejected. This is because n is composed of large primes p and q. If a number has small primes (factors less than the size of p and q) then it could not be n and the trial password could not be the true password. The authors of EKE give reasons in their paper as to why an attacker even after substituting his own n for RSA modulus will not be able to mount a dictionary attack. We disagree and present just such an attack.

A querying attacker, impersonating Alice, sends Bob his specially generated RSA modulus n and the password encrypted public key X. Of course the attacker does not know the password P, but just sends a random number X instead of the encrypted RSA exponent $P(e)$. Bob unknowingly, retrieves e as $P^{-1}(X)$. For the present, assume that we some how know e has 3 as a factor, that is, $e = 3k$ for some k. Now when Bob picks a random number R and transmits $P(R^e)$, we know it is also equivalent to $P(R^{3k})$. The attacker can now try different candidate passwords P' from the dictionary, decrypt $P'^{-1}(P(R^{3k}))$ into $(R^{3k})'$ and see if this number is a cubic residue. If it is not then the attacker can reject that password P'. About one ninth (we justify this later) of the passwords on average will result in decryptions which are cubic residue $\bmod(n = pq)$, and the remaining nine tenths of the passwords can be rejected. Another session will allow us to further reject nine tenths of the remaining set of possible passwords. So at a logarithmic rate, in the number of sessions, we can narrow the space of valid passwords down to one.

Now of course, there is no way to know a priori if a given random number e is of the form $3k$; but we don't need to know if we are willing to repeat the above scenario with multiple random numbers e by picking different random X. So now we similarly pick e and try to narrow the passwords down to one. If this e does not have 3 as a factor then we will quickly see that no candidate of passwords remain which have always yielded a cubic residue. Hence we know this e does not have 3 as a factor, assuming of course that the correct password is in the constructed dictionary. We try again with a different random e, until we see that one and only one password consistently yields cubic residues. Since one out of 3 random e will have factors of 3, we need on average 3 e before finding the desired e.

To verify if a given decryption $(R^e)'$ is a cubic residue $\bmod n$ we have to verify if it is a cubic residue $\bmod p$ and $\bmod q$. We can do this if we have picked p such that $p-1$ has 3 as a factor and q such that $q-1$ has 3 as a factor. We could then raise the given number to $\frac{p-1}{3} \bmod p$ and it will result in 1 if e has a 3 as a factor because $(R^e)'^{\frac{p-1}{3}} \bmod p \equiv (R^{k3})^{\frac{p-1}{3}} \bmod p \equiv (R^k)^{p-1} \bmod p$ which by Fermats theorem is congruent to $1 \bmod p$. We similarly proceed with $\bmod q$. The existence of residues and their numbers are stated more formally later.

This is not just a theoretical attack, but quite a practical one which can rapidly lead to the discovery of the password. In fact only tens of sessions with a legitimate RSA-EKE user are sufficient to break the password of that user. Lets say that after each try one ninth of the passwords in the dictionary still remain as possibly valid while the rest are discarded. In order to narrow down the password down to one we need on average i queries such that $1 = (\frac{1}{9})^i W$, where W is the size of the dictionary. For a dictionary of size one million we need about 6 sessions and since we need on average 3 random e, we need on average 18 sessions to crack the password!

There is an obvious, but unsatisfying, defense to such an attack. A machine can limit the number of consecutive unsuccessful tries and upon reaching that limit force a change in the password. Querying attackers can try to evade such requirements by interleaving their unsuccessful sessions with successful sessions carried by legitimate users. Since the number of sessions needed for our attack is in the tens, setting an overall limit for unsuccessful attempts may cause users to rebel against the frequent password change and write down the password. Another serious side effect is that denial of service attacks now become much easier.

3.6 Neutralizing the Attack

Even if an attack is successful it does not mean it is fatal. An attack can often be side stepped by adding special

conditions as has been the case with the RSA cryptosystem where attacks have been side stepped by requiring the two prime numbers to be strong primes. Let us see how we can try to side step our attack on RSA-EKE.

Modification 1: Legitimate users should not respond to e with low factors.

Countermove: When e can only have higher factors, a greater number of random e have to be tried by the querying attacker, but for each e a far greater subset of the passwords can be rejected. Nevertheless, legitimate users can check as high as necessary to make the number of random e to be tried so large as to make the attack impractical. Factors which do not make the attack impractical will be referred to as low factors.

However, a simpler attack is now possible without recourse to number theory because the user is telling the querying attacker that e has low factors by rejecting the session. An attacker sends a random X instead of $P(e)$ to Alice. Alice will retrieve $e = P^{-1}(X)$ and reject communication if an e has low factors. The attacker can use this information to directly rule out all passwords by similarly decrypting X by a candidate password P' to form $e' = P'^{-1}(X)$. If an e' has low factors then the candidate password P' can be rejected.

Modification 2: The legitimate user will still respond to e with low factors, but with bogus numbers instead of $P(R^e)$.

Countermove: These modifications may stop the querying attacker from conducting attacks to get information, but now an eavesdropping attacker can simply eavesdrop on legitimate sessions to rule out passwords. The above modifications dictate that the legitimate users must always avoid low factors. So an eavesdropping attacker collects legitimate $P(e)$ transfers from sessions and tries to decrypt with passwords from a dictionary. If any passwords yield a decryption of e with low factors, then that trial password will be rejected. As always, this will continue with more sessions until one valid password remains.

Modification 3: To avoid failures of modification 2, legitimate users can send e so as not to leak any information. To pick e, we start with a random number X then check if X is an e without low factors. If not then we try the next number and the next until a suitable e without low factors has been found. Transfer the number X to Bob. At the other end Bob would also start with X and search for e in the same way. Now an eavesdropper cannot get any information that would allow him to rule out passwords because each number has an equal likelihood of occurring.

Countermove: This will successfully solve the problem with the previous modifications. However, there is a more basic problem when e with low factors are not allowed. The problem is that now an attacker can always gets a response and can test for cubic non-residue to validate passwords instead of cubic residue. When modification 3 is in place, a querying attacker sends a random X to Bob. Bob will derive an e from X such that e does not have any low factors. Finally, Bob answers the attacker with $P(R^e)$. The attacker can test this for cubic non-residue to validate passwords. We discuss this further in section 3.8.

3.7 Facts from Number Theory

All simple modifications, as in the previous section, will eventually fail. We will see this more clearly after giving some number theory results. This will also formalize some aspects of the attack that we have only discussed informally.

Theorem 1 *Let p be a prime such that $p-1$ has integer d as a factor. The congruence $x^d \equiv a \bmod p$ has a solution and a is a dth power residue when $a^{\frac{p-1}{d}} \equiv 1 \bmod p$, else the congruence does not have a solution and a is a dth power non-residue when $a^{\frac{p-1}{d}} \not\equiv 1 \bmod p$.*

Corollary 1 *The number of dth power residues $\bmod p$ is equal to $\frac{p-1}{d}$.*

The theorem and corollary are basic results in solving congruence equations and are proved in many texts on number theory [5]. In fact Theorem 1 is a special case of the general result: Let p be a prime, m an integer, and $d = \gcd(m, p-1)$, the congruence $x^m \equiv a \bmod p$ has a solution and a is an mth power residue when $a^{\frac{p-1}{d}} \equiv 1 \bmod p$, else the congruence does not have a solution and a is the mth power non-residue when $a^{\frac{p-1}{d}} \not\equiv 1 \bmod p$. Theorem 1 is the special case when d is equal to m.

Theorem 2 *Let p and q be primes such that $p-1$ and $q-1$ have d as a factor and $n = pq$. The congruence $x^d \equiv a \bmod n$ has a dth power residue if and only if $a^{\frac{p-1}{d}} \equiv 1 \bmod p$ and $a^{\frac{q-1}{d}} \equiv 1 \bmod q$.*

Proof: By theorem 1, $x^d \equiv a \bmod p$ and $x^d \equiv a \bmod q$ both have solutions. A corollary of the chinese remainder theorem [3] states that $x \equiv a \bmod p$ and $x \equiv a \bmod q$ have a solution if and only if $x \equiv a \bmod pq$. Hence we see that theorem 2 is correct and $x^d \equiv a \bmod pq$ will have a solution if and only if $x^d \equiv a \bmod p$ and $x^d \equiv a \bmod q$ have a solution.

Corollary 2 *The number of dth power residue $\bmod n$ which are prime to n equals $\frac{p-1}{d}\frac{q-1}{d}$.*

Proof: Again by the chinese remainder theorem there is a one to one mapping between $\mathbf{Z_n}$ and the cartesian product $\mathbf{Z_p} * \mathbf{Z_q}$. Hence the number of solutions in the congruence equation $\bmod n$ is the product of the number of solutions in the congruence equations $\bmod p$ and $\bmod q$.

We see that in our example attack d was equal to 3 and $p-1$ and $q-1$ have 3 as a factor. Also the number of cubic residue equals $\frac{p-1}{3}\frac{q-1}{3}$ or $\frac{\varphi(n)}{3^2}$. Since about $\frac{1}{9}$ numbers will pass as cubic residues, only $\frac{1}{9}$ passwords of the dictionary will remain as possible candidates, the other passwords will be rejected because their decryption did not yield a cubic residue.

3.8 The Attack on RSA-EKE is Successful

In a countermove to modification 3, we mentioned that there is no escape even when we try to avoid low factors in e. This is so because Theorem 2 gives an if and only if condition for a dth power residue. If the condition is not met then we know the number is a dth power non-residue or a cubic non-residue in our example. So if we know that e cannot have 3 as a factor then we check for cubic non-residue and about $\frac{9}{10}$ passwords remain as possible candidates. This is larger than $\frac{1}{9}$, but does not change the logarithmic nature of the speed with which password choices are narrowed down to one.

EKE is caught between a rock and a hard place. If it allows low factors then we test for cubic or more generally dth power residue $\bmod n$ as described in section 3.5 and reject a subset of passwords. If on the other hand, EKE does not allow low factors then we test for cubic or dth power non-residue and still rule out subsets of passwords. We could substitute testing for cubic residue by testing for cubic non-residue in the attack described in section 3.5. The flaw is fatal. Only radical revisions of the protocol can possibly avoid these kinds of attack. In Figure 2 we restate the complete attack in an algorithmic form.

Although we have used 3 as an example of a low factor repeatedly, we could have used 5, 7, or higher factors. The higher the factor, the larger the subset of passwords that are rejected at each iteration, but offsetting that is the fact that more random e will have to be tried before finding e with the intended factor. One can choose the appropriate factor to maximize the speed of the attack. Most likely, there are also other number theoretic relationships which may be exploited to discover passwords.

```
Create n = pq, such that p − 1 and q − 1 have factor 3.
Repeat until 1 password is consistently valid
{
    Pick a random e ( by picking a random X )
    ValidPasswords ← dictionary
    Until 0 or 1 password is consistently valid
    {
        Attacker sends Bob: X
        Bob replies: P(R^e) where e = P^{-1}(X)
        For all passwords P' in ValidPasswords
        {
            (R^e)' = P'^{-1}(P(R^e))

            If (EKE allows 3 as a factor of e) then
              If ( (R^e)' is a cubic residue )
              i.e. If ( ((R^e)')^{(p-1)/3} ≡ 1 mod (p, q) )
                Keep P' in ValidPasswords
              else
                Remove P' from ValidPasswords

            If (EKE does not allow 3 as a factor of e) then
              If ((R^e)' is a cubic non-residue
              i.e. If ( ((R^e)')^{(p-1)/3} ≢ 1 mod (p, q) )
                Keep P' in ValidPasswords
              else
                Remove P' from ValidPasswords
        }
    }
}
```

Figure 2. Algorithm for Attack on RSA-EKE

4 The Diffie Hellman Version

4.1 Brief Description of the Diffie Hellman Key Exchange

In the Diffie Hellman exchange, a prime p and a primitive element or generator g are publicly known. In order to agree on a key, Alice and Bob both will generate random numbers R_A and R_B between 0 and $p-1$ and calculate $g^{R_A} \bmod p$ and $g^{R_B} \bmod p$ respectively. These quantities are publicly exchanged. Now Alice raises the received value g^{R_B} by her chosen random value R_A to form $g^{R_A R_B} \bmod p$. Bob similarly raises the received value g^{R_A} to his chosen random value R_B to form $g^{R_A R_B}$.

Now both Alice and Bob are in possession of a common value or key $g^{R_A R_B} \bmod p$. No one else can calculate this key because the discrete log problem is considered to be a difficult problem for suitably large p. Even though g, p, $g^{R_A} \bmod p$ and $g^{R_B} \bmod p$ are publicly known, no one can discover R_A or R_B in reasonable time. The Diffie Hellman key exchange is a very elegant public key distribution scheme, but unfortunately it suffers from a practical problem: lack of authentication. Even exchanging a password to authenticate, after communication has been established, is not enough because a 'man in the middle' who has established communications with both Alice and Bob would be able to read messages and relay without detection. The EKE version avoids this problem.

4.2 Description of the Diffie Hellman Encrypted Key Exchange (DH-EKE)

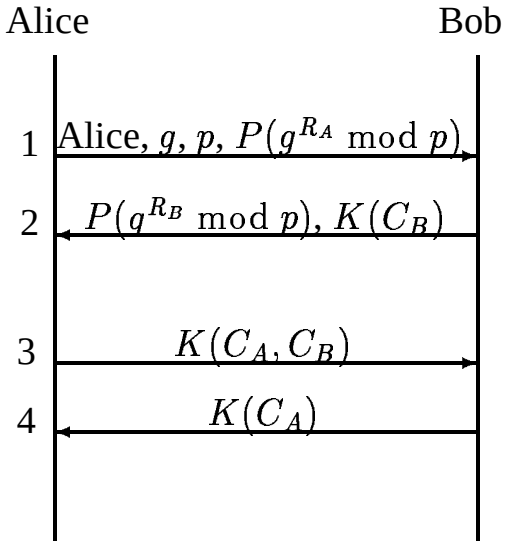

Figure 3. EKE using the Diffie Hellman Key Exchange

In step one (Figure 3) Alice sends Bob $g^{R_A} \bmod p$, encrypted by the shared password P. Alice also sends her name, the prime p, and generator g unencrypted to Bob. In step 2 Bob sends Alice $g^{R_B} \bmod p$, encrypted by the password P. Both can decrypt the received number and raise it to their random number to generate $g^{R_A R_B} \bmod p$ which will be used to derive the session key K. Now both can proceed with standard challenge/response steps to authenticate the session key. We do not detail these further steps but just show that this can begin in step 2 with Bob sending a challenge C_B encrypted by K. An eavesdropper cannot validate password guesses because R_A and R_B are random hence $g^{R_A} \bmod p$ and $g^{R_B} \bmod p$ will also appear random. Furthermore, even if they are correctly guessed, there is no way to calculate the session key $K = g^{R_A R_B} \bmod p$. A 'man in the middle attack' is also foiled because without knowing the password the attacker cannot read and relay messages.

4.3 Number Theoretic Attack on DH-EKE

A querying attacker, pretending to be Alice, sends his g, p and a random number X instead of $P(g^{R_A} \bmod p)$ since he does not know the password P. Bob upon receiving the transmission, generates a random number R_B and calculates $g^{R_B} \bmod p$ and sends it encrypted by P to the attacker. At this point it is not clear if the attacker can use any information to rule out passwords because R_B appears random, $g^{R_B} \bmod p$ appears random and its further encryption, $P(g^{R_B} \bmod p)$ appears to be random. However, we will show that by judicious choosing of g and p the attacker can discover the password.

Instead of sending the g and p in clear, he chooses to send g^d, and p where d is a low prime (e.g. 2, 3,..) and $p-1$ has factor d. Now when Bob raises that to an exponent the result is $a = (g^d)^{R_B} \bmod p$ which is the same as $g^{R_B d} \bmod p$. This number, a, is obviously a dth power residue and by reason of Theorem 1, $a^{\frac{p-1}{d}} \equiv 1 \bmod p$ if $p-1$ has a factor d. This can also be seen more directly: $(g^{R_B d})^{\frac{p-1}{d}} \bmod p$ is equal to $(g^{R_B})^{p-1} \bmod p$ and by Ferrmat's theorem it is congruent to $1 \bmod p$. When the attacker receives the password encrypted $P(g^{R_B d} \bmod p)$, he tries decrypting it with different candidate passwords and raises the decrypted number to $\frac{p-1}{d} \bmod p$. If the result is not 1 then that password is rejected. By corollary 1 we know that $\frac{p-1}{d}$ numbers out of $p-1$ numbers will be dth power residues, hence $\frac{1}{d}$ numbers on average will be congruent to $1 \bmod p$ when raised to $\frac{p-1}{d}$. At each session the possible space of password is reduced to $\frac{1}{d}$ and the space of valid passwords will be narrowed to one at a logarithmic rate.

4.4 Attack Avoidance

This attack is unavoidable if the attacker is allowed to use his choice of g^d and p. Fortunately, this attack can be avoided by checking for these specific choices and not allowing them. The number theory enthusiast will have noticed by now that g^d is not a generator. To find a generator g it is necessary and sufficient to check that $g^{\frac{p-1}{k}} \not\equiv 1 \bmod p$ for all factors k of $p-1$. This necessary condition is violated by the attacker because the element g^d satisfies $g^{d\frac{p-1}{d}} \equiv 1 \bmod p$ and hence is not a primitive root or generator of p.

The EKE authors wisely advise that it is desirable that g be a primitive root, but we have shown it to be an absolute necessity. The suggestion that $p-1$ be of form kp' where p' is a large prime is convenient for testing generators. There has to be an agreement among the parties as to what range of values are allowed for k, so p' can be retrieved, tested for primality and also so that g can be verified as a generator. The users cannot be lax about checking all these conditions in order to save on computations. It is not enough to check that p is a prime or that p' is a prime, but also that g is a true generator of p.

4.5 Attack on Half-Encrypted DH-EKE

The authors of EKE suggest further variations in the protocols, in particular, encryption of one of the steps may be omitted as long as the other step is encrypted. Exactly

which one can be omitted depends upon the particular protocol and cryptosystem. For the DH-EKE, the authors state that it is possible to omit encryption of one of the exponentials in either step 1 or step 2. They give the example of message $g^{R_A} \bmod p$ in step 1 being sent unencrypted without danger, although they do caution against an attacker substituting 0 for the exponent and causing $g^{R_A R_B} \equiv 1 \bmod p$ which gives away the session key K. We claim that encryption of both messages is a necessity unless challenges are specially typed.

4.5.1 Attack with the First Message Unencrypted

We present attacks on both variants of the half- encrypted DH-EKE. We first deal with the message in the first step being sent unencrypted. A querying attacker, pretending to be Alice, can pick R_A and send $g^{R_A} \bmod p$ to Bob. Upon receiving the message Bob will generate R_B and raise g^{R_A} to R_B to form $g^{R_A R_B} \bmod p$. This is used to generate the session key K, and then in step 2 Bob sends $P(g^{R_B} \bmod p)$ and $K(C_B)$ to the attacker.

The attacker can now calculate candidate values for K. For each password P' in the dictionary, the attacker can decrypt and get candidate $(g^{R_B})'$, but the attacker knows R_A since he generated it and can raise $(g^{R_B})'$ by R_A to get candidates $(g^{R_A R_B})'$ which gives K'. If there is a redundancy in the challenge $K(C_B)$ sent in step 2, then we can discover the password. The authors of EKE mention that the redundancy in the challenge/response part of the protocol may be the result of the challenge being typed or there being a checksum. Let us say the challenge is typed by one bit to indicate the sender of the challenge. Say the typed bit is 0 if A is the sender and it is 1 if B is the sender. In step 2, B in reality does not just send $K(C_B)$ but $K(C_B, 1)$. The attacker can then reject all passwords P' which result in K' yielding decryptions of $C_B, 0$. Half of the passwords can be rejected and at a logarithmic rate the passwords will be discovered. The authors of EKE [1] present a similar attack on the half-encrypted version of ElGamal-EKE.

4.5.2 Attack with the Second Message Unencrypted

We are now ready to prove the second part of our claim that both messages have to be encrypted by attacking the variant with only the first message encrypted. Alice initiates a session with Bob by sending $P(g^{R_A} \bmod p)$, but the active attacker intercepts the communication and pretends to be Bob. The active attacker in step 2 sends the message $g^{R_B} \bmod p$ in clear, for a R_B he picked. Since the attacker is picking R_B, he can form candidate K' for different P' by first decrypting $P(g^{R_A} \bmod p)$ into candidate $(g^{R_A} \bmod p)'$. Then raise that exponential by R_B to form $(g^{R_A R_B})'$ which gives candidate K'.

However, now the attacker does not have an apparent way to validate K' even assuming there is a redundancy in the challenge/response messages. The attacker can try an indirect approach. He cannot send a legitimate typed challenge $K(C_B, 1)$ because he does not know K. Instead the attacker just sends a random number X to Alice. If she accepts the message then the attacker knows that under K, X decrypts to ${C_B}', 1$. If she rejects it then the message X decrypts to ${C_B}', 0$. The attacker can now use this information to reject half of the passwords P' and with more sessions narrow the choice down to one password!

Perhaps this second version of the half-encrypted DH-EKE can benefit from moves like those in section 3.6 to avoid revealing information and we would have to give countermoves. The only good move left for Alice is to try to avoid revealing information useful for validation of K by always replying and not hanging the session in the middle. The full EKE protocol specifies that Alice should send $K(C_A, C_B)$ to the attacker.

Now what should Alice use for C_B? Should she use $K^{-1}(X)$ for C_B or just another random number? Either way the password can be discovered. If $K^{-1}(X)$ is used then the attacker has to just try all candidates K' resulting from passwords P' and eventually one P' or K' will result in a match between $K'^{-1}(X)$ and $K'^{-1}(C_B)$ which is the valid K and hence the valid password. If Alice uses a random number for C_B then the attacker still tries all candidate K' to see if $K'^{-1}(X)$ and $K'^{-1}(C_B)$ match. However, now none will match and the attacker will know that Alice has just sent a bogus C_B trying to obfuscate his attack because the X sent by him really decrypts to $({C_B}', 0)$ under K instead of $(C_B, 1)$. Knowing this, the attacker continues to reduce the space of valid passwords at logarithmic speed.

There is one way out of the attack on the second version of half-encrypted DH-EKE. The attacks is not possible if Alice uses heavily typed challenge/response instead of one bit being 0 or 1. If Alice is identified by 64 bits of 0 and Bob by 64 bits of 1s then Alice can reject X sent by the attacker right away without a $K(C_A, C_B)$ response because its extremely improbable that decryption of $K^{-1}(X)$ will yield $({C_B}', 111...1111)$. This thwarts the attack. So the overall claim should be modified to: A)the half-encrypted DH-EKE does not work with step 1 unencrypted and B)the half-encrypted DH-EKE with step 2 unencrypted does not work unless one is very careful of the typed messages and redundancy in the challenge steps. For example, a random heavily typed message is no good and will not stop the attacks outlined in this section.

5 The ElGamal Version

5.1 Brief Description of the ElGamal Cryptosystem

Although the ElGamal system is based on the Diffie Hellman key exchange, it is a full fledged public key cryptosystem. The values of prime p, generator g, and $g^{R_A} \bmod p$ are public where as R_A is kept secret by Alice. To send an encrypted message m, Bob generates a pair of numbers $y1$ and $y2$ to transfer to Alice. First Bob picks a random number k, less than p, and creates $y1 \equiv g^k \bmod p$. He also generates $y2 \equiv m(g^{R_A})^k \bmod p$. Alice can retrieve the message m by $m = y2(y1^{R_A})^{-1} \bmod p$.

5.2 Description of ElGamal-EKE

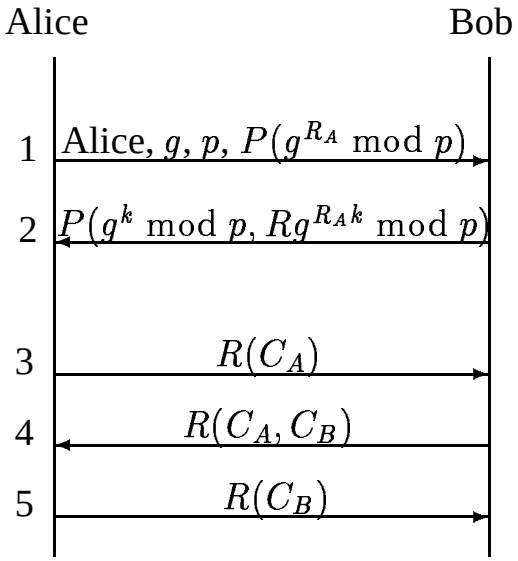

Figure 4. EKE using the ElGamal Cryptosystem

In step 1 (Figure 4) Alice sends Bob $P(g^{R_A} \bmod p)$. Alice also sends her name, the prime p, and generator g in the clear. In step 2 Bob sends Alice $P(g^k \bmod p)$, where R is the session key to be used and Alice can retrieve R by first decrypting the password encryption, and performing the previously mentioned operations to retrieve message m or in this case, session key R. Alice can now send a challenge message and both continue to authenticate the session key R, but we do not give further details here.

An eavesdropper cannot validate password guesses because R_A, R and k are random hence $g^{R_A} \bmod p$, $g^k \bmod p$ and $Rg^{R_A k} \bmod p$ are random and no information is leaked under encryption by password P.

5.3 Number Theoretic Attack on the ELGamal-EKE

ElGamal-EKE is susceptible to the similar kind of number theoretic attack as the DH-EKE. So it is also absolutely necessary in the ElGamal-EKE to guarantee that g is a generator of p. Here is how the attack would proceed: The querying attacker, pretending to be Alice, sends g^d, p and also X instead of $P(g^{R_A} \bmod p)$ since he does not know the shared password P. Of course, $p-1$ is picked with d as a factor. Upon receiving the messages, Bob first calculates $(g^d)^k \bmod p$ or $g^{kd} \bmod p$. He also decrypts the random value X into $Y = P^{-1}(X)$. Then he sends the attacker, in step 2, $P(g^{kd} \bmod p, RY^k \bmod p)$.

The attacker now tries to decrypt the message from Bob using candidate passwords P'. If the decryption is correct then $(g^{kd})^{\frac{p-1}{d}}$ would always be congruent to 1. Hence, any trial passwords resulting in $(g^{kd})^{\frac{p-1}{d}} \not\equiv 1 \bmod p$ can be rejected. This continues with other sessions until the space of valid passwords is narrowed to one at the proverbial logarithmic rate.

5.4 Attack on the Half Encrypted ElGamal-EKE

We claim that messages in both step 1 and step 2 have to be encrypted in the ElGamal-EKE. The authors of EKE show why at least step 2 requires the message to be password encrypted or else an attacker can use redundancy in the R encrypted challenge, $R(C_A)$, from Alice to validate candidates for R and in turn password candidates. The EKE authors assume that the challenges are typed to avoid cut and paste types of attacks or contain other types of redundancy.

We now show why the message in step 1 also has to be encrypted. A querying attacker, pretending to be Alice, picks R_A and sends $g^{R_A} \bmod p$ in the clear. Bob upon receiving it responds with $P(g^k \bmod p, Rg^{R_A k} \bmod p)$ in step 2. The attacker can now form candidate session keys R'. First he decrypts Bob's message by using candidate passwords P'. This results in candidates $(g^k)'$ and $(Rg^{R_A k})'$. Since the attacker picked R_A, he knows it and can raise $(g^k)'$ to R_A to form $(g^{R_A k})'$. R' then is simply $\frac{(Rg^{R_A k})'}{(g^{R_A k})'} \bmod p$.

However, there is no direct way to validate R', but we can use a similar indirect validation scheme as we did in section 4.5.2 for DH-EKE. The attacker cannot send a legitimate typed challenge $R(C_A, 0)$ because he does not know R. Instead the attacker just sends a random number X to Bob. If he accepts the message then the attacker knows that under K, X decrypts to ${C_A}', 0$. If he rejects it then the message X decrypts to ${C_A}', 1$. The attacker can now use this information to validate candidate R' and in turn pass-

words P'. After few sessions, the shared password would be revealed.

As mentioned in the section 4.5.2, lightly typed challenges allow candidate passwords to be rejected. Only heavily typed challenges, for example, a fixed string of 64 bits can block the attack in this section and in section 4.5.2.

6 Gong-Lomas-Needham-Saltzer Protocols

These protocols too use a combination of public key and secret key cryptosystems to create secure protocols which are also resilient to dictionary attacks. Random numbers, called confounders, are used like one-time pads to block guessing attacks. Many of these protocols are based on a server S mediating on behalf of clients Alice and Bob. These protocols can thus use well chosen public keys stored at the server and do not need to generate different public keys for each session. These protocols appear to be secure against known active and passive attacks, including dictionary attacks.

However, two versions of the protocol generate new public keys for each session and are vulnerable to dictionary attacks mounted by an attacker. Direct Authentication Protocol allows Alice and Bob to establish a well-chosen session key without the aid of a server. Secret Public Key Protocol also allows Alice and Bob to establish a well-chosen session key with the aid of a server even if neither Alice nor Bob remembers the server's public key. We will not describe these protocols in detail, but describe them enough to understand how attacks can be mounted and relate it to the experience in previous sections. The description of these protocols [4] are generic and do not give an example usage of a concrete public key system (e.g. RSA). Hence, we are forced to create our own variants using the RSA public key system, and our attacks are specific to the RSA variants of the protocols. The use of other public key systems may or may not be vulnerable to attacks. In creating the RSA variants we have been careful to follow the authors advice not to allow any information leakage. Our attacks are not based on any form of redundancy in the message or leakage as described in section 3.3.

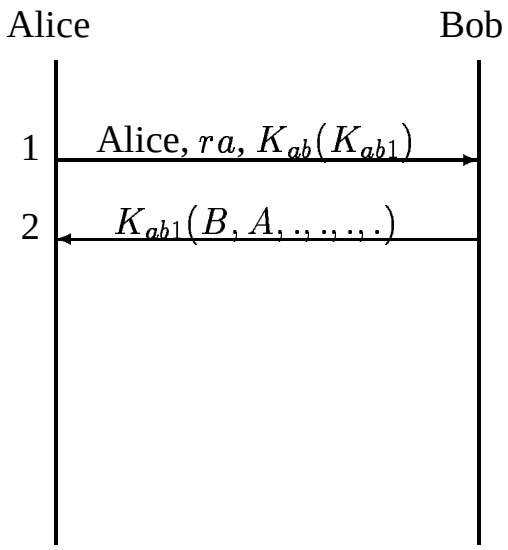

Figure 5. Generic Version of the Direct Authentication Protocol

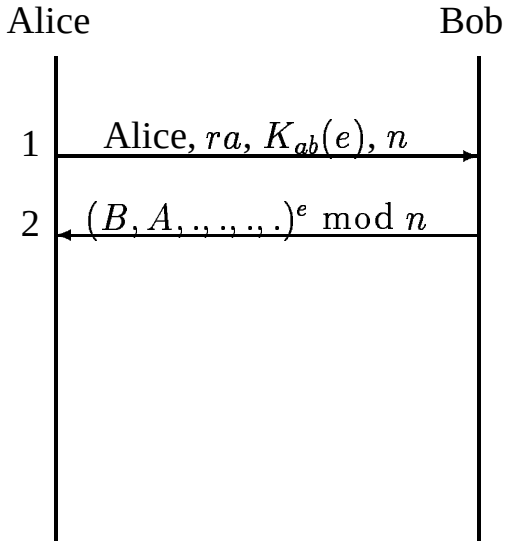

Figure 6. A RSA Variant of the Direct Authentication Protocol

6.1 Direct Authentication Protocol (RSA Variant)

Alice and Bob already share a secret password K_{ab} and they wish to establish a well-chosen session key. Alice has to generate a new public key K_{ab1} for every session.

Figure 5 lists only the first two steps of the protocol because the others are not necessary for the attack. In step 1, Alice sends Bob a random number ra and a public key K_{ab1} encrypted by the shared password K_{ab}. Bob responds with his name, B, and Alice's name, A, and other random numbers not listed in the figure; all encrypted by the public key K_{ab1}.

This is still not enough for us to describe the attack. In order to describe the attack we present a RSA variant of the Direct Authentication Protocol in Figure 6. In step 1 we see that n has to be sent in the clear and only the RSA exponent e is sent encrypted by shared password K_{ab}. We saw in section 3.5 that sending n in the clear is necessary because an

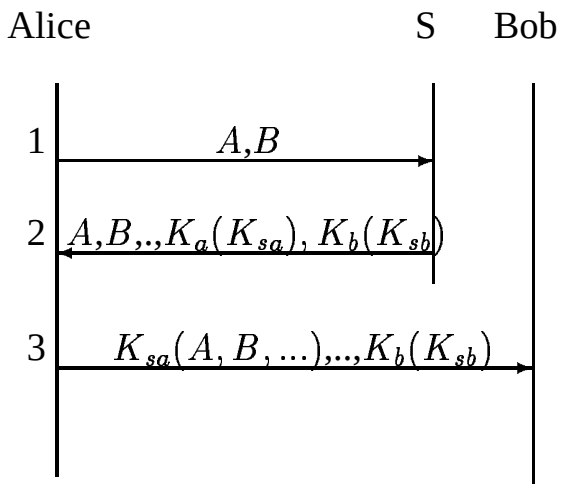

Figure 7. Generic Version of the Secret Public Key Protocol

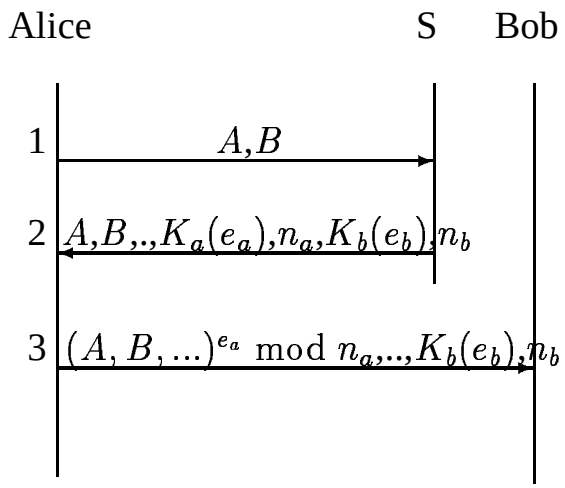

Figure 8. A RSA Variant of the Secret Public Key Protocol

attacker can decrypt $P(n)$ by different passwords and those decryption which yield numbers with small primes can be rejected. In response Bob sends $(B, A, ..)$ and other random numbers raised to the RSA exponent $e \bmod n$. Now we are ready to present the attack.

A querying attacker, impersonating Alice, sends Bob (Alice, ra, X, n). Since the attacker is generating n, he knows the prime factors of n. Knowing p and q, he is in possession of $\varphi(n)$ and in turn can find the decrypting exponent d for any e. Since the attacker does not know the secret password K_{ab}, he sends a random X instead of $K_{ab}(e)$. Bob unwittingly decrypts X into $e = K_{ab}^{-1}(X)$. Furthermore in step 2, Bob sends the attacker $(B, A, ., ., .)^e \bmod n$. The attacker can now also for each candidate P' decrypt X into $e' = P'^{-1}(X)$. Since the attacker knows $\varphi(n)$, he can create the decrypting exponent d' for each e'.

Now the attacker uses each decrypting exponent d' and decrypts $((B, A, ., ., .)^e)^{d'} \bmod n$. If the resulting (B', A') equal the true (B, A) then that password P' is the secret password K_{ab}!

6.2 Secret Public Key Protocol (RSA Variant)

Alice shares a secret K_a with the server S and Bob shares a secret K_b with S. Alice and Bob wish to establish a well-chosen session key. Since Alice and Bob do not remember the server's public key, the server will send to them the public key.

Figure 7 shows the generic version of the Secret Public Key protocol. In step 1 Alice sends the server its request that it wants a session with Bob. In step 2 the server responds to Alice with the name of the session parties A,B, the public key K_{sa} encrypted by the secret password K_a, and the public key K_{sb} encrypted by the secret password K_b. In step 3 Alice sends Bob the message $(A, B, ., ., .)$ encrypted by the public key K_{sa} and also relays the public key K_{sb} encrypted by the secret password K_b. There are 7 steps in all to completely understand the protocol, but we do not describe these and refer the reader to the original paper [4].

This is still not enough for us to describe the attack. In order to describe the attack we present a RSA variant of the Secret Public Key protocol in Figure 8.

An active attacker, acting as the server S, blocks Alice's communication with the real server and sends Alice: A, B, X_a, n_a, X_b, and n_b. Instead of the true n_a, the attacker generates and uses his own n_a whose prime factors he knows. Knowing p and q, he is in possession of $\varphi(n_a)$ and in turn can find the decrypting exponent d for any e. Since the active attacker does not know the secret password K_a, he sends a random X_a instead of $K_a(e_a)$. Alice unwittingly decrypts X_a into $e_a = {K_a}^{-1}(X_a)$. Furthermore in step 3, Alice sends Bob $(A, B, ., ., .)^{e_a} \bmod n_a$. This is overheard by the attacker. The attacker can now also for each candidate P' decrypt X_a into $e' = P'^{-1}(X_a)$. Since the attacker knows $\varphi(n_a)$, he can create the decrypting exponent d'_a for each e'_a.

Now the attacker uses the decrypting exponed d'_a and decrypts $((A, B, ., ., .)^{e_a})^{d'_a} \bmod n_a$. If the resulting (A', B') equal the true (A, B) then that password P' is the secret password K_a. Similar steps would be necessary to discover K_b.

7 Why the Attacks are Possible

We need to ask why the number theoretic attacks were successful at all. In all cases the attack was possible because some basic assumption of the non-EKE version of the cryptosystem or key exchange was violated. In the case of the Diffie Hellman key exchange and the ElGamal cryptosystem, the assumption that a generator or primitive element was used was violated. Fortunately, in these two systems, such violations can detected. However, in the case of the RSA version of EKE such checks are not possible or are very difficult at best (EKE authors discuss interactively querying to verify the choice of e).

One solution is for everyone to use agreed upon public key constants. This option is also discussed in the EKE paper [1]. Assuming these constants (e.g. g, p, n, e) came from a trustworthy source then we can have confidence that the underlying assumptions of the cryptosystem or key exchange are satisfied. However, these public key constants can be targets for intense cryptanalytic efforts since breaking them would in essence reveal the passwords of all users who use them. Furthermore, users have to be in possession of the correct versions of these long public key constants.

In the case of half encrypted versions of EKE, picking one of the exponentials in either steps gives us enough information to validate trial passwords, assuming redundancy in the challenge messages. To avoid exploitation of the typed message for validating passwords, large typed messages could be used in some instances. The successful use of large typed messages is dependent upon the exact steps of the challenge/response protocol.

In the case of Direct Authentication and Secret Public Key protocols the attacker was able to substitute his own n to eventually discover the secret password. As in the case with other Gong-Tomas-Needham-Saltzer protocols, if widely agreed public key constants were used then this problem could be avoided.

In the design of secure password protocols, encryption is useful in protecting against guessing attacks. However, encryption should not be done at the expense of public key system verification. By verification we mean that both parties must be able to verify, for example, that they are engaged in a bona fide RSA system exchange and not a reasonable facsimile. All assumptions that are verifiable in the unencrypted use of the public key system must also be verifiable when public key is exchanged encrypted by a shared password among users. And if we are unable to verify an assumption like $n = pq$, as previously seen in the RSA versions of EKE and Gong-Lomas-Needham-Saltzer protocols, then we should not use the RSA versions of these and similar protocols. Note that verifying the assumptions of a public key system is a necessary condition for security but not a sufficient condition. Other conditions like eliminating information leakage or keeping leakage in check are also necessary for security.

8 Conclusion

We have presented number theoretic attacks against all versions of EKE. In one case, the RSA-EKE version, the attack is unavoidable unless the protocol is radically modified. We also showed that it is an absolute necessity that g be verified to be a generator of p in the Diffie Hellman and the ElGamal versions. Furthermore we have shown how an attacker is able to discover the secret passwords in the RSA variants of Direct Authentication and Secret Public Key protocols.

We explored why such attacks are successful against seemingly secure password protocols and discovered that assumptions of the public-key system are not verified or are unverifiable in these password protocols. We also presented attacks against half encrypted versions of EKE and showed that it is generally necessary that both steps be encrypted and that half encrypted EKE versions not be used unless special care is taken in typing of the challenge messages.

9 Acknowledgements

I am indebted to Stuart Haber, Sivaram Rajagopalan, and Ramarathnam Venkatasen for many insightful discussions. I also want to thank Reshma Patel and Moti Yung for many valuable comments and corrections.

References

[1] S. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. *Proceedings IEEE Computer Society Symposium on Research in Security and Privacy*, pages 72–84, May 1992.

[2] S. Bellovin and M. Merritt. Augmented Encrypted Key Exchange. *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 224–250, November 1993.

[3] T. Cormen, C. Leiserson, and R. Rivest. *Introduction To ALGORITHMS*. The MIT Press, Cambrige, MA, 1995.

[4] L. Gong, T. M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting Poorly Chosen Secrets from Guessing Attacks. *IEEE Jornal on Selected Areas in Communications*, 11(5):648–656, June 1993.

[5] A. Hurwitz and N. Kritikos. *Lectures on Number Theory*. Springer-Verlag, New York, NY, 1986.

[6] D. V. Klein. Foiling the Cracker: A Survey of, and Implications to, Password Security. *Proceedings of the USENIX UNIX Security Workshop*, pages 5–14, August 1990.

[7] T. M. A. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham. Reducing Risks from Poorly Chosen Keys. *Proceedings of the 12th ACM Symposium on Operating System Principles, ACM Operating Systems Review*, 23(5):14–18, December 1989.

[8] S. Patel. Information Leakage in Encrypted Key Exchange. *to be published in Proceedings of the DIMACS Workshop on Network Threats*, 1997.

[9] R. L. Rivest, A. Shamir, and L. Adleman. A Methord for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.