Analysis of EAP-SIM Session Key Agreement

Sarvar Patel
Lucent Technologies
sarvar@lucent.com

Abstract: EAP-SIM specifies a mechanism for mutual authentication and session key agreement using the GSM-SIM and by proposing enhancement to the GSM authentication procedures. Unfortunately, as we show it does not succeed in its goal of providing 128 bit security from the current 64 bit security of GSM. Furthermore, it does not provide session independence between different sessions. For the first problem, we are able to provide solutions, but the second problem does not seem solvable in practice.

1. EAP-SIM Brief Description

We only concentrate on a main part of EAP-SIM which is the mutual authentication and session key agreement protocol. There are other aspects of EAP-SIM mechanism which we do not analyze, for example, fast re-authentication, identity protection, algorithm specifications for MAC, PRF, etc.

1) C→A: R_c
2) A→C: R_1,R_2,R_3, MAC_k[…, R_1,R_2,R_3,R_c]
3) C→A: MAC_k[…,$SRES_1$,$SRES_2$,$SRES_3$]

We are only showing partial steps and parts of the protocol useful for our analysis, for the full description please see [1]. Also our notation is differs from [1] for simplicity purposes. In step 1, the client C sends to the authenticator A a random challenge R_c. In the next step A responds with a list of random challenges, up to 3 of them; The protocol allows 1 t o3 random challenges to be sent, but we specify it to be 3 in our description because that is suppose to be the strongest case. These challenges are gotten from the GSM system from the triplets where each triplet consists of RAND, SRES, K_c. The RAND is a 128 bit number used with a root key K_i (upto 128 bits) to generate a 64 bit key K_c and a 32 bit value SRES. Thus in our case 3 triplets are sent providing 3 RAND, SRES, and K_c values

A also sends the MAC of its 3 random numbers and R_c. There are other information that is MACed for example type, but we leave out the details. In order to MAC, a key is needed. This is generated by MK=SHA[…,K_{c1},K_{c2},K_{c3},R_c,…]. This is then fed to a PRF to generate K_aut and other keys; we will call K_auth just k for brevity in this document. When C receives the string of RANDs and the MAC, it verifies the MAC using first the K_i to generate the K_c vector and in turn using them to generate MK and the mac key k. In

step 3, C MACs the SRES vector using the key k and sends it to A. A in turn verifies the MAC using key k and the SRES vector it had received from the GSM system.

2. Analysis

GSM keys K_c are only 64 bits long and thus directly using them would not provide greater security. EAP-SIM tries to use a vector of them to create session keys. Since now, if 3 K_c's are used the overall keysize can be 192 bits long and one can hope that we have achieved 128 bit security.

2.1 Only 64 bits security

Unfortunately, as we show that the security reached is only 64 bits and not 128 bits even when 3 K_c's are used. One simple way to attack an encryption key is to guess its value and proceed with the protocol. If the session key is 128 bit strong then the probability of success should be around $1/2^{128}$. However, we will show how to proceed with a guessing attack with success probability $1/2^{64}$. This should not happen with a secure 128 bit key and indicates the security is actually still 64 bits.

We show how an authenticator impersonator A' can carry a full conversation with a client. First A' makes a guess at a (R',K_c') pair. Since the K_c is only 64 bits long, the probability of guessing correctly is $1/2^{64}$. Then in step 2, A' sends to C the string RAND',RAND',RAND', MACk[…,RAND',RAND',RAND',R_c]. A' is able to correctly compute the MAC because by correctly guessing the key K_c', its able to create MK=SHA[…,K_c',K_c',K_c',R_c,..] and mac key k from it.

The client will verify the MAC and accept the RAND values. Furthermore it will generate session keys and carry a full conversation with the impersonator A'.

2.1.2 Solutions

There are couple of solutions to this problem which one hopes may address these specific concerns. The first solution is to simply have the client insist that the RANDs have to be different from each other. For example, if 2 RANDs are used then insist that R1 not equal R2. If three RANDS are used then insist that R1 not equal R2 and R1 not equal R3, and R2 not equal R3. This way the adversary will have to guess values at least two different points and that means the success probability will be less than $1/2^{128}$.

However, this does not help if only one RAND is used in the vector. Another solution is to insist that SRES also be part of the MK calculation. In that case the adversary would have to guess not only the values of K_c but also SRES. Even if only one RAND is used, this means the adversary has to guess at 96 bits which is better than 64 bits.

Of course, the two solutions can be combined, so that in the case where more than 1 RAND is used we would hope to get 128 bits of security and where only one RAND is used we still hope to get to 96 bits of security.

2.2 Lack of Session Independence

The protocol, however, suffers from a serious deficiency that its sessions are not independent. If the K_c values from one of the sessions is compromised then an adversary can use them to carry fraudulent conversations with the client. We can see this is the case because the K_c values are dependent on the RANDs and not on the random challenge from the Client. Thus if the RAND and K_c vectors are ever compromised, an attacker can use them directly. The client who receives the RANDs would not know that these were compromised values and would go ahead with the protocol, verify the MAC and accept the attacker as a legitimate network.

A secure mutually authenticated key session agreement protocol should not have this problem. It might be suggested that all one has to do is to make sure that the Kc vector is never compromised. This is not practical to assume in a general protocol because even if the authenticator is trusted today, its not clear that it can be trusted for all future times which is what would be needed to make sure that a past Kc vector is never used again. Secondly, even if one trusts an authenticator for one type of conversation in one location, its not clear that one would want to trust for other types of conversations. But a compromised K_c could effect all future conversations at all locations.

Another way to see that session independence is important is to look at why are we using different session keys at all for encryption and message authentication? A part of the answer is to not expose lots of plaintext/ciphertext but a large part of the answer is what we just stated about independence.

Furthermore, if one could trust that the K_c vectors will never be compromised, then one never has to generate more than one K_c vector. Just use the same K_c vector for different sessions, since the R_c challenge from the client will be different we can assume that the message encryption and authentication session keys will be different. There is no reason to generate different triplets at all. Actually, there is no reason to even generate one triplet, one could just share the root key K_i itself if one is assured that it will not be compromised. Then using the 128 bit K_i one could use a very strong protocol to guarantee mutually authenticated session key agreement protocol.

2.2.1 Solutions

There is actually no solution to this problem as long as one is working with GSM triplets as the fundamental source of keying. The above solutions to strengthen the 64 bit to 128 bits are of no use in creating session independence, and it’s hard to see how one could do it easily.

One approach possible but not practical is for the client to store all the past RAND vectors its seen and to make sure that they are not repeated. If it sees that a previous RAND vector is being re-used it should satisfy the protocol but abort before using the session keys to actually encrypt anything. In this case, if the real network ever repeated a

previous vector accidently, that vector would be used up and at the next try the network would use a fresh vector which will be different. The chance of a vector being accidently repeated is negligible because an individual RAND is 128 bits.

Since its not practical to store all past values, perhaps the client can store the most recent n RAND values and make sure they are not repeated. This may give some partial protection in practice. Actually the whole RAND vector doesn't need to be stored, just a part of the RAND, for example 32 bits can be stored and looked for repeats; this way many previously used values can be checked for repeats. Of course, if a previous value was repeated then the client should make sure that value is always checked for repetition in the future and never allowed.

3. Conclusions

The EAP-SIM protocol does not provide 128 bits of protection and we gave an attack showing that only 64 bits of protection is provided. We also gave solutions to address this problem. Secondly, EAP-SIM does not provide session independence and this is a serious deficiency. Unfortunately, there does not seem to be a solution which can fully eliminate this problem as long as the keying material is based on the GSM triplets.