Comparison and Implementation of Password-based Public Key Cryptographic Protocols

Xuesong Chen

Johns Hopkins University
Information Security Institute

MSSI Project (Non-Credit)

January 24, 2008

Project Advisor: Professor Fabian Monrose

Academic Advisor: Professor Gerald M. Masson

Abstract

This document presents the current status and a comparison of three cryptographic protocols - SRP6, AMP2 and PAK-Z+, which are candidates of IEEE P1363.2 standards for Password-Based Public-Key Cryptographic Techniques. The comparison includes four aspects: 1. Security Features 2. Security Proofs and Assumptions 3. Performance 4. Flexibility and Optimization. This document also discusses implementation issues of AMP-2 in Simnet.

1 Introduction

Password is a simple and popular method for human authentication. Password authentication protocols have wide range of applications in practice because memorable password has been used in authentication for a long time in history and no trusted third party is needed. But traditional password authentication protocols are normally based on challenge-response techniques and vulnerable to offline dictionary attack so that users have to remember long and complex password and even change password frequently. In 1990s some password authentication protocols using public-key cryptography were proposed and they have some promising security features. Then IEEE formed the P1363.2 group to standardize password-authenticated key agreement schemes. SRP, AMP and PAK-Z are the most popular 3 candidates and SRP-6, AMP-2 and PAK-Z+ are the latest versions. All of them are claimed to have similar important and interesting security features. By comparing their differences and discussing implementation issues, advantages and disadvantages of each protocol are revealed.

2 Comparison

2.1 Security Features

All of the 3 protocols are claimed to have the following security features:
1. A passive attacker can not learn anything about user's password and session keys from protocol messages (Zero-Knowledgeness). So these protocols are resistant to offline dictionary attack which is effective to many old password-based authentication techniques and protocols such as Kerberos.
2. Resistant to active attacker: even if the attacker can forge or alter

messages, no information about password or session key is revealed in a way other than one password guessing per protocol session(which is detectable).
3. Because of 1 and 2, low entropy(human-memorable) password can used in practice without worrying about dictionary attack.
4. Perfect Forward Secrecy: If user's long-term password is compromised, an attacker can not deduce session keys used in previous sessions; if a session key is compromised, an attacker can not deduce user's long-term password (even if all the session messages are captured by the attacker).
5. Resistant to server compromise: even if server's password file (which stores password verifiers) is captured by the attacker, the attacker can not impersonate the user without expensive dictionary search.

2.2 Security Proofs and Assumptions

Although all the 3 protocols are claimed to be resistant to active attacks, only PAK-Z+ has formal theoretical security proofs in the random oracle model. The security model used by PAK-Z+ security proofs is a revised version of the model built by M. Bellare, et, all in paper "Authenticated key exchange secure against dictionary attacks". PAK-Z+ is claimed to be the only protocol "proven secure that is being considered for the IEEE P1363.2 standard". Since the security proofs for PAK-Z+ is relatively new(2005), and actually PAK-Z+ is a revised version of PAK-Z, which also was claimed to have formal theoretical security proofs and later proven to be broken(not resistant to server compromise), it is still too early to fully trust the security proofs. It is easy to show that all the 3 protocols can be reduced to Diffie-Hellman so that they are resistant to passive attacks and have perfect forward secrecy. While the security of SRP and AMP only relies on good hash functions and Diffie-Hellman problem, PAK-Z+ requires additional signature scheme which should be existentially unforgeable against adaptive chosen message attacks.

2.3 Performance

All the 3 protocols can be implemented in 1.5 - 2 round(3-4 messages) communication for mutual authentication. SRP-6(4 messages) needs a hash function and modular addition, subtraction, multiplication and power computation. AMP-2(4 messages) requires 5 different hash functions and modular addition, multiplication and inversion computation. PAK-Z+(3 messages) asks for 6 different hash functions, XOR, modular multiplication,

inversion and power computation and signature generation and verification.

	Add	Sub	Mul	Inv	Xor	Pow	Hash	Sig/G	Sig/V
SRP-6 Client	1	1	2	0	0	3	4	0	0
AMP-2 Client	2	0	2	1	0	2	5	0	0
PAKZ+ Client	0	0	1	1	2	2	6	1	0
SRP-6 Server	1	0	1	0	0	3	3	0	0
AMP-2 Server	0	0	2	0	0	3	4	0	0
PAKZ+ Server	0	0	1	0	1	2	3	0	1

From the above table, we can see SRP-6 and AMP-2 should have similar performance on the server side and AMP-2 may have slight advantage on the client side because it requires less expensive modular power computation. The performance of PAK-Z+ depends on underlying signature scheme. If the computational time for the signature scheme is comparable to modular power, then its overall performance in slow networks should be better than the other 2 because it requires fewer message rounds.

2.4 Flexibility and Optimization

SRP-6 is the most simple and flexible one of the 3 protocols. It does not require modular inversion and signature scheme which may be expensive. Without adding more message round, it can also be used in the case that Diffie-Hellman parameters are sent by server so that client side software does not need to store them (need to check their validity) and only needs user's password. This is a very useful feature when client software is installed on public computers. If we want to add this feature to AMP-2 and PAK-Z+, additional message round is necessary because their first message sent by server depends on their first message sent by client. Thus message ordering in AMP-2 and PAK-Z+ can not be optimized like SRP-6.

3 Implementation Issues of AMP-2

SRP is the most widely accepted protocol of the three. Many versions of telnet and ftp using SRP have been available for years and 2 IETF RFCs

have been approved. PAK-Z is also implemented in Lucent Technologies' Plan9 operating system and it requires complex existentially unforgeable signature scheme. Finally I choose to implement AMP-2 in Simnet.

There are several implementation issues:

1. Diffie-Hellman parameter generation:
   I use the "DHParametersGenerator" class in bouncy castle library to generate group parameters(P, Q, G_). P is a safe prime such that P=2Q+1. Q is a large prime. G_ is a generator of the group Z{P, *}. But AMP-2 needs generator G of the subgroup Z{P, Q}. We can simply compute G=square(G_) or let G=4(because 4 must be a quadratic residue: 4=square(2)). At present 1024-bit P is considered large enough to be secure. For convenience in my implementation demo I use 256-bit P.

2. Password verifier generation:
   According to AMP-2, we first compute u=h0(user_id, password), then compute verifier "v=G^u mod p" and store it on server. The verifier is not plaintext-equivalent to password.

3. Choosing hash functions:
   I use SHA1 to generate 5 different hash functions.
   h0(input)=SHA1(0, input)
   h1(input)=SHA1(1, input)
   h3(input)=SHA1(3, input)
   h4(input)=SHA1(4, input)
   h5(input)=SHA1(5, input)

4. Random number generation:
   Both server and client need to generate a random number. These random numbers should be selected from Z{Q} which is [0, ..., Q-1]. For convenience I use 160-bit random number([0, ..., 2^160 - 1]) whose bit length is equal to SHA1 digest size. As long as the range for random numbers is large enough and cryptographic PRNG(not statistical PRNG) is used for unpredictability, there should be no problem.

5. Constraint checks:
   We should check if the numbers (m and Mu) exchanged in the first message round are in the group Z{P, Q}. First we check if they are in the range [1, P-1], then check if (m or Mu).modPow(Q, P)==1. If yes, then they

are elements in group Z{P, Q}. It is because Z{P, Q} is a prime-order group, each element in the group (other than 1) is a generator of the group.

6. Minor mistake in the paper:

There is a minor mistake in the paper about server's computing the number "Beta". Beta should be (m*G)^y mod P, not (m'*G)^y mod P because Beta should be equal to the number "Alpha" computed by client which is equal to G^((x+1)*y)

user$[id, \pi]$ on A		server$[id, \nu = g^u \bmod p]$ on B
$x \leftarrow_R \mathbb{Z}_q$		
$m \leftarrow g^x \bmod p$		
	$\xrightarrow{C, m}$	
$u \leftarrow h_0(C, \pi)$		$y \leftarrow_R \mathbb{Z}_q$
$e \leftarrow h_1(C, S, m)$		$e \leftarrow h_1(C, S, m)$
		$m' \leftarrow m^e \bmod p$
$w \leftarrow (xe + u)^{-1}(x + 1) \bmod q$		$\mu \leftarrow (m'\nu)^y \bmod p$
	$\xleftarrow{\mu}$	
$\alpha \leftarrow \mu^w \bmod p$		$\beta \leftarrow (m'g)^y \bmod p$
$k_1 \leftarrow h_3(C, S, m, \mu, \alpha)$		$k_1' \leftarrow h_3(C, S, m, \mu, \beta)$
	$\xrightarrow{k_1}$	
		abort if $k_1 \neq k_1'$
$k_2' \leftarrow h_4(C, S, m, \mu, \alpha)$		$k_2 \leftarrow h_4(C, S, m, \mu, \beta)$
	$\xleftarrow{k_2}$	
abort if $k_2 \neq k_2'$		
$K \leftarrow h_5(C, S, m, \mu, \alpha)$		$K' \leftarrow h_5(C, S, m, \mu, \beta)$

Fig. 4. AMP2

	Client		Server
1.		$\xrightarrow{I}$	(lookup s, v)
2.	$x = H(s, I, P)$	$\xleftarrow{s}$	
3.	$A = g^a$ $u = H(A, B)$	$\xrightarrow{A}$ $\xleftarrow{B}$	$B = 3v + g^b$ $u = H(A, B)$
4.	$S = (B - 3g^x)^{a+ux}$		$S = (Av^u)^b$
5.	$M_1 = H(A, B, S)$	$\xrightarrow{M_1}$	(verify M_1)
6.	(verify M_2)	$\xleftarrow{M_2}$	$M_2 = H(A, M_1, S)$
7.	$K = H(S)$		$K = H(S)$

Table 5: SRP with refinements (SRP-6)

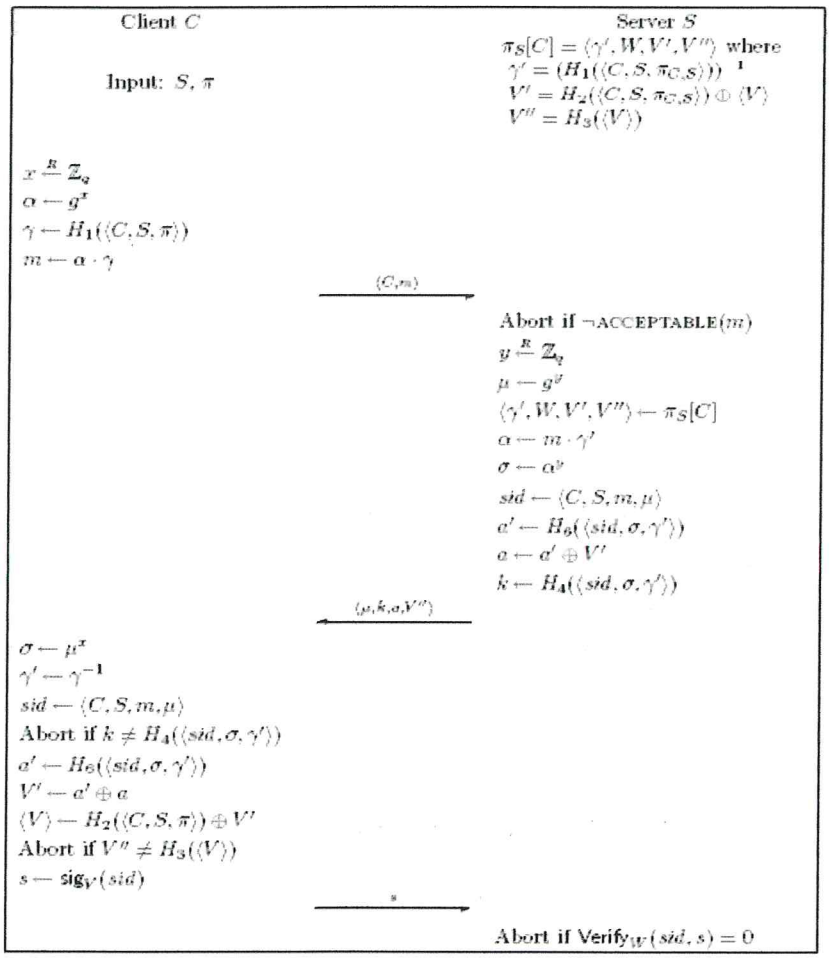

Figure 1: PAK-Z+ Protocol. Partner ID for C is $pid_C = S$, and partner ID for S is $pid_S = C$. Shared session key is $sk = H_5(\langle sid, \sigma, \gamma' \rangle)$.

References

[1] M. Bellare, D. Pointcheval and P. Rogaway, Authenticated key exchange secure against dictionary attack," In *Eurocrypt 2000*, LNCS 1807, pp.139-155, 2000.

[2] T.Wu. The secure remote password protocol. In *1998 Internet Society Network and Distributed System Security Symposium*, pages 97-111, 1998.

[3] T.Wu. A real-world analysis of Kerberos password security. In *Proceedings of the 1999 Network and Distributed System Security Symposium*, February 1999.

[4] T. Wu, SRP6: Improvements and refinements to the secure remote password protocol," Unpublished document, October 2002.

[5] T. Kwon, Authentication and key agreement via memorable password, In *ISOC Network and Distributed System Security Symposium*, February 2001.

[6] T. Kwon, Revision of AMP in IEEE P1363.2 and ISO/IEC-11770-4, In *IEEE P1363.2 contributions*, August 2005.

[7] P. MacKenzie. The PAK Suite. DIMACS Tech Report 2002-46.

[8] P. MacKenzie. PAK-Z+. In *IEEE P1363.2 contributions*, August 2005.

AMP2Utils.java

```
package AMP2;

import java.math.*;
import java.security.*;
import org.bouncycastle.crypto.generators.*;
import org.bouncycastle.crypto.params.*;
import org.bouncycastle.crypto.digests.*;

public class AMP2Utils
{
	//256-bit Diffie-Hellman parameters:
	//P is a safe prime(P=2*Q+1)
	//Q is also a prime
	//G is a generator for sub-group Z{P,Q}, not Z{P,*}
	private static final String Pstr =
"68063943173371350091984194315129643911781388786018533262812784472290160135299";
	private static final String Qstr =
"34031971586685675045992097157564821955890694393009266631406392236145080067649";
	private static final String G_str =
"67490252040511775937893777330345664343851025284249827911823376234405325978 96";

	public static final BigInteger P = new BigInteger(Pstr);
	public static final BigInteger Q = new BigInteger(Qstr);
	public static final BigInteger G = new BigInteger(G_str).modPow(new BigInteger("2"), P);
	//public static final BigInteger G = new BigInteger("4");

	public static final boolean checkMembership(BigInteger n) {
		// check if 1 <= n <= P-1
		if(n.compareTo(BigInteger.ZERO) <= 0 ||
		  n.compareTo(P) >= 0)
		{
			return false;
		}

		//check if n is a quadratic residue (mod P)
		if(n.modPow(Q, P).equals(BigInteger.ONE))
		   return true;
		else
		   return false;
	}

	//user and server id string
	public static final String C="username";

	public static final String S="servername";

	//get "different" hash algorithms
	public static final SHA1Digest get_h(byte n) {
		SHA1Digest h = new SHA1Digest();
		h.update(n);
		return h;
	}
```

```
        //generate and check safe prime
        public static void main(String[] args)
        {
                /*DHParametersGenerator dhpg = new DHParametersGenerator();
                dhpg.init(256,100,new SecureRandom());
                DHParameters dhp = dhpg.generateParameters();

                System.out.println("P = "+dhp.getP().toString());
                System.out.println("Q = "+dhp.getQ().toString());
                System.out.println("G = "+dhp.getG().toString());*/

                /*if( P.isProbablePrime(100) &&
                   Q.isProbablePrime(100) &&
                   P.shiftRight(1).compareTo(Q)==0)
                {
                        System.out.println("safe prime ok!");
                }*/

                //compute u
                //user password
          String Pi="password";
          SHA1Digest h0 = get_h((byte)0);
          byte[] C_buf = C.getBytes();
          h0.update(C_buf, 0, C_buf.length);
          byte[] Pi_buf = Pi.getBytes();
          h0.update(Pi_buf, 0, Pi_buf.length);
          byte[] u_buf = new byte[20];
          h0.doFinal(u_buf, 0);
          BigInteger u = new BigInteger(1, u_buf);
          //compute v = G^u mod P
          BigInteger v = G.modPow(u, P);
          System.out.println("v="+v);
        }
}
```

AMP2Client.java

```
package AMP2;

import java.util.*;
import java.math.*;
import simnet.*;
import java.security.*;
import org.bouncycastle.crypto.digests.*;


/** AMP2 Client */
public class AMP2Client extends Application implements Pluggable {
```

```
//user password
String Pi="password";

public void customInit() {
        // nothing to do
}

private void print(String msg) {
        System.out.println(msg);
}

public void setupAMP2Session(String server, Integer port) throws Exception {

        print("C: connect to server: "+server+" port: "+port);
        Socket sock=((TCP)node.getTransport(Simnet.PROTO_TCP)).createSocket(this);
  sock.connect(sim.lookup(server), port.intValue());
  print("C: connected");

  //compute x and m, send C and m to server
  //160 bits for x (x should be less than Q(255 bits))
  BigInteger x = new BigInteger(160, new SecureRandom());
  BigInteger m = AMP2Utils.G.modPow(x, AMP2Utils.P);
  byte[] m_buf = m.toByteArray();
  print("C: send C="+AMP2Utils.C+" m="+m);
  sock.send(AMP2Utils.C);
  sock.send(new ByteArrayWrapper(null, m_buf));

  //compute u
  SHA1Digest h0 = AMP2Utils.get_h((byte)0);
  byte[] C_buf = AMP2Utils.C.getBytes();
  h0.update(C_buf, 0, C_buf.length);
  byte[] Pi_buf = Pi.getBytes();
  h0.update(Pi_buf, 0, Pi_buf.length);
  byte[] u_buf = new byte[20];
  h0.doFinal(u_buf, 0);
  BigInteger u = new BigInteger(1, u_buf);
  //compute e
  SHA1Digest h1 = AMP2Utils.get_h((byte)1);
  h1.update(C_buf, 0, C_buf.length);
  byte[] S_buf = AMP2Utils.S.getBytes();
  h1.update(S_buf, 0, S_buf.length);
  h1.update(m_buf, 0, m_buf.length);
  byte[] e_buf = new byte[20];
  h1.doFinal(e_buf, 0);
  BigInteger e = new BigInteger(1, e_buf);

  //compute w
  BigInteger w = x.multiply(e).add(u).mod(AMP2Utils.Q);
  w = w.modInverse(AMP2Utils.Q);
  w = w.multiply(x.add(BigInteger.ONE)).mod(AMP2Utils.Q);

  //receive Mu
  Object obj=null;
  while(!((obj=sock.recv()) instanceof ByteArrayWrapper));
  BigInteger Mu = new BigInteger(((ByteArrayWrapper)obj).data);
  print("C: receive Mu="+Mu);
```

```
if(!AMP2Utils.checkMembership(Mu))
{
        print("C: abort: invalid value of Mu");
        sock.close();
        return;
}

//compute Alpha
BigInteger Alpha = Mu.modPow(w, AMP2Utils.P);
print("C: compute Alpha="+Alpha);

//compute and send k1 to server
SHA1Digest h3 = AMP2Utils.get_h((byte)3);
h3.update(C_buf, 0, C_buf.length);
h3.update(S_buf, 0, S_buf.length);
h3.update(m_buf, 0, m_buf.length);
byte[] Mu_buf = Mu.toByteArray();
h3.update(Mu_buf, 0, Mu_buf.length);
byte[] Alpha_buf = Alpha.toByteArray();
h3.update(Alpha_buf, 0, Alpha_buf.length);
byte[] k1 = new byte[20];
h3.doFinal(k1, 0);
ByteArrayWrapper k1_buf = new ByteArrayWrapper(null, k1);
print("C: send k1="+k1_buf);
sock.send(k1_buf);

//compute k2_
SHA1Digest h4 = AMP2Utils.get_h((byte)4);
h4.update(C_buf, 0, C_buf.length);
h4.update(S_buf, 0, S_buf.length);
h4.update(m_buf, 0, m_buf.length);
h4.update(Mu_buf, 0, Mu_buf.length);
h4.update(Alpha_buf, 0, Alpha_buf.length);
byte[] k2_ = new byte[20];
h4.doFinal(k2_, 0);

//receive k2
while(!((obj=sock.recv()) instanceof ByteArrayWrapper));
byte[] k2 = ((ByteArrayWrapper)obj).data;
print("C: receive k2="+obj);

//abort if k2 != k2_
if(!Arrays.equals(k2, k2_))
{
        print("C: abort: invalid k2");
        sock.close();
        return;
}

//compute the final K
SHA1Digest h5 = AMP2Utils.get_h((byte)5);
h5.update(C_buf, 0, C_buf.length);
h5.update(S_buf, 0, S_buf.length);
h5.update(m_buf, 0, m_buf.length);
h5.update(Mu_buf, 0, Mu_buf.length);
h5.update(Alpha_buf, 0, Alpha_buf.length);
```

```
            byte[] K = new byte[20];
            h5.doFinal(K, 0);
            print("C: compute K="+new ByteArrayWrapper(null, K));

            sock.close();
        }

        /** Tell our thread we are done, and close the socket if we are not being replaced. */
        public boolean prePlugout(Object replacement) {

                return true;
        }

}
```

AMP2Server.java

```
package AMP2;

import java.util.*;
import java.math.*;
import simnet.*;
import java.security.*;
import org.bouncycastle.crypto.digests.*;

/** AMP2 Server */
public class AMP2Server extends Application implements Pluggable {

        //password verifier stored on server
        BigInteger v = new
BigInteger("2431828292770315687280483852183375874548338617663206672935895162814049956536
2");
        //BigInteger v = new
BigInteger("1269026692982964509270654692984017773755491556014598773878354334039822251175
6");

        public void customInit() {
                // nothing to do
        }

        private void print(String msg) {
                System.out.println(msg);
        }

        private int port=8888;

        public void startAMP2Server(Integer port)
        {
                this.port=port.intValue();
                new Thread(this).start();
```

```
	}

	public void run()
	{
		try
		{
			ServerSocket
ssock=((TCP)node.getTransport(Simnet.PROTO_TCP)).createServerSocket(this);
			ssock.bind(port);
		ssock.listen(3);

		while(true)
		{
			Socket sock=ssock.accept();
			print("S: new session request");
			Object obj=null;

			//receive C, m
			while(!((obj=sock.recv()) instanceof String));
			String C = (String)obj;
			if(!AMP2Utils.C.equals(C))
			{
				print("S: abort: invalid user="+C);
				sock.close();
				continue;
			}
			while(!((obj=sock.recv()) instanceof ByteArrayWrapper));
			BigInteger m = new BigInteger(((ByteArrayWrapper)obj).data);
			print("S: receive C="+C+" m="+m);
			if(!AMP2Utils.checkMembership(m))
	{
		print("S: abort: invalid value of m");
		sock.close();
		continue;
	}

			//compute y
			//160 bits for y (y should be less than Q(255 bits))
		BigInteger y = new BigInteger(160, new SecureRandom());

		//compute e
		SHA1Digest h1 = AMP2Utils.get_h((byte)1);
		byte[] C_buf = AMP2Utils.C.getBytes();
		h1.update(C_buf, 0, C_buf.length);
		byte[] S_buf = AMP2Utils.S.getBytes();
		h1.update(S_buf, 0, S_buf.length);
		byte[] m_buf = m.toByteArray();
		h1.update(m_buf, 0, m_buf.length);
		byte[] e_buf = new byte[20];
		h1.doFinal(e_buf, 0);
		BigInteger e = new BigInteger(1, e_buf);

		//compute m_
		BigInteger m_ = m.modPow(e, AMP2Utils.P);
```

```
//compute Mu and send it to client
BigInteger Mu = m_.multiply(v).mod(AMP2Utils.P).modPow(y, AMP2Utils.P);
print("S: send Mu="+Mu);
byte[] Mu_buf = Mu.toByteArray();
sock.send(new ByteArrayWrapper(null, Mu_buf));

//compute Beta
BigInteger Beta = m.multiply(AMP2Utils.G).mod(AMP2Utils.P).modPow(y,
AMP2Utils.P);
print("S: compute Beta="+Beta);

//compute k1_
SHA1Digest h3 = AMP2Utils.get_h((byte)3);
h3.update(C_buf, 0, C_buf.length);
h3.update(S_buf, 0, S_buf.length);
h3.update(m_buf, 0, m_buf.length);
h3.update(Mu_buf, 0, Mu_buf.length);
byte[] Beta_buf = Beta.toByteArray();
h3.update(Beta_buf, 0, Beta_buf.length);
byte[] k1_ = new byte[20];
h3.doFinal(k1_, 0);

//receive k1
while(!((obj=sock.recv()) instanceof ByteArrayWrapper));
        byte[] k1 = ((ByteArrayWrapper)obj).data;
        print("S: receive k1="+obj);

        //abort if k1 != k1_
        if(!Arrays.equals(k1, k1_))
{
           print("S: abort: invalid k1");
           sock.close();
           continue;
}

//compute k2 and send it to client
SHA1Digest h4 = AMP2Utils.get_h((byte)4);
h4.update(C_buf, 0, C_buf.length);
h4.update(S_buf, 0, S_buf.length);
h4.update(m_buf, 0, m_buf.length);
h4.update(Mu_buf, 0, Mu_buf.length);
h4.update(Beta_buf, 0, Beta_buf.length);
byte[] k2 = new byte[20];
h4.doFinal(k2, 0);
ByteArrayWrapper k2_buf = new ByteArrayWrapper(null, k2);
print("S: send k2="+k2_buf);
sock.send(k2_buf);

//compute the final K
SHA1Digest h5 = AMP2Utils.get_h((byte)5);
h5.update(C_buf, 0, C_buf.length);
h5.update(S_buf, 0, S_buf.length);
h5.update(m_buf, 0, m_buf.length);
h5.update(Mu_buf, 0, Mu_buf.length);
h5.update(Beta_buf, 0, Beta_buf.length);
byte[] K = new byte[20];
```

```
                    h5.doFinal(K, 0);
                    print("S: compute K="+new ByteArrayWrapper(null, K));

                        sock.close();
                }//while
                //ssock.close();

        }//try
        catch(Exception excp){}


            }//run method


            /** Tell our thread we are done, and close the socket if we are not being replaced. */
            public boolean prePlugout(Object replacement) {

                    return true;
            }

}
```

amp2_test.script

```
# load a network with no latency or bandwidth
load networks/test.net
start all
usp
slp all all 0 0 0 0

# load a AMP2Server on JHU
s JHU
plug in AMP2.AMP2Server 100
startAMP2Server 8888

# load a AMP2Client on MIT
s MIT
plug in AMP2.AMP2Client 101
setupAMP2Session JHU 8888
quit
```