EAP-TLS original

usertype string;

const req, shd, ccs: string;

const mastersecret, clientfinished, serverfinished, keyexpansionclient, keyexpansionserver, accesslevel, success: string;

hashfunction h, prf;

macro m=( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R));

macro m1= {h(m)}sk(I);

macro ms= prf(PMS, mastersecret,ni, nr);

macro mc= ( {CerI, pk(I)}sk(CA), {PMS}pk(R));

macro m2= prf (ms, serverfinished, h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc, {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc)}sk(I), ccs));

macro m3= prf (ms, clientfinished, h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc, {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc)}sk(I),prf (ms, serverfinished, h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc), {h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc )}sk(I), ccs)));

macro clientkey=prf(ms, keyexpansionclient, ni, nr);

macro serverkey= prf(ms, keyexpansionserver, ni, nr);

protocol eaptls (I,R,CA)

{

role I

{

const SID, CerI, CerR,PMS: Data;

fresh ni: Nonce;

var nr:Nonce;

send\_1 (I, R, SID, ni);

recv\_4 (R, I, SID, nr, {CerR,pk(R)}sk(CA), req, shd);

send\_5 (I, CA, I);

recv\_6 (CA, I, {I, {CerI, pk(I)}pk(I)}sk(CA));

send\_7 (I, R, {CerI, pk(I)}sk(CA), {PMS}pk(R));

send\_8 (I,R , {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R) )}sk(I));

recv\_9 (R, I, ccs);

recv\_10 (R, I, prf (ms, serverfinished, h(m, m1)));

match (m2, prf (ms, serverfinished, h(m, m1)));

send\_11 (I, R, prf (ms, clientfinished, h(m, m1, m2)));

recv\_12 (R, I, {accesslevel}serverkey);

send\_13 (I, R, {success}clientkey);

claim\_I1 (I, Secret, CerR);

claim\_I2 (I, Secret, CerI);

claim\_I3 (I, Secret, PMS);

claim\_I4 (I, Secret, ms);

claim\_I5 (I, Secret, clientkey);

claim\_I6 (I, Secret, serverkey);

claim\_I7 (I, Nisynch);

claim\_I8 (I, Niagree);

}

role R

{

const SID, CerI, CerR, PMS: Data;

fresh nr: Nonce;

var ni:Nonce;

recv\_1 (I, R, SID, ni);

send\_2 (R, CA, R);

recv\_3 (CA, R, {R, {CerR,pk(R)}pk(R)}sk(CA));

send\_4 (R, I, SID, nr, {CerR,pk(R)}sk(CA), req, shd);

recv\_7 (I, R, {CerI, pk(I)}sk(CA), {PMS}pk(R));

recv\_8 (I,R , {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R) )}sk(I));

match(m1, {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R))}sk(I));

send\_9 (R, I, ccs);

send\_10 (R, I, prf (ms, serverfinished, h(m , m1)));

recv\_11 (I, R, prf (ms, clientfinished, h(m, m1, m2)));

match (m3, prf (ms, clientfinished, h(m, m1, m2)));

send\_12 (R, I, {accesslevel}serverkey);

recv\_13 (I, R, {success}clientkey);

claim\_R1 (R, Secret, CerR);

claim\_R2 (R, Secret, CerI);

claim\_R3 (R, Secret, PMS);

claim\_R4 (R, Secret, ms);

claim\_R5 (R, Secret, clientkey);

claim\_R6 (R, Secret, serverkey);

claim\_R7 (R, Nisynch);

claim\_R8 (R, Niagree);

}

role CA

{

const CerI, CerR: Data;

recv\_2 (R, CA, R);

send\_3 (CA, R, {R, {CerR, pk(R)}pk(R)}sk(CA));

recv\_5 (I, CA, I);

send\_6 (CA, I, {I, {CerI, pk(I)}pk(I)}sk(CA));

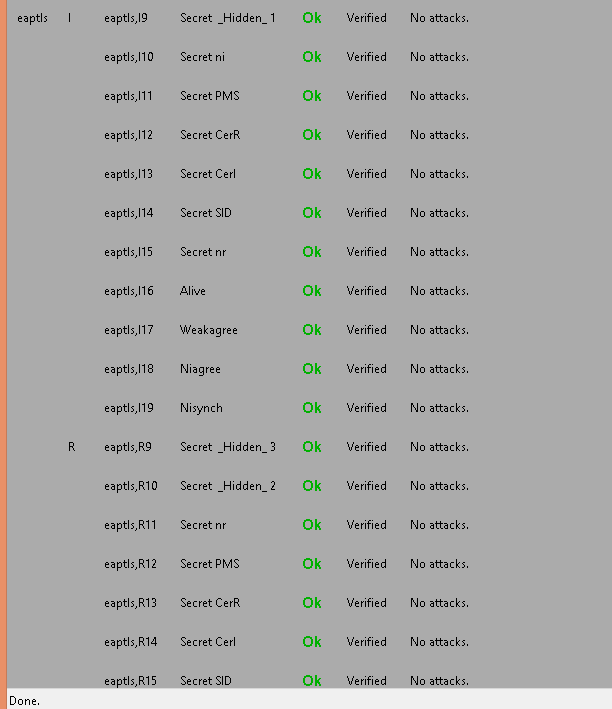
claim\_CA1 (CA, Secret, CerR);

claim\_CA2 (CA, Secret, CerI);

}

}





EAP-TLS modified

usertype string;

const req, shd, ccs: string;

const mastersecret, clientfinished, serverfinished, keyexpansionclient, keyexpansionserver, accesslevel, success: string;

hashfunction h, prf;

macro m=( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R));

macro m1= {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R))}sk(I);

macro ms= prf(PMS, mastersecret,ni, nr);

macro mc= ( {CerI, pk(I)}sk(CA), {PMS}pk(R));

macro m2= prf (ms, serverfinished, h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc, {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc)}sk(I), ccs));

macro m3= prf (ms, clientfinished, h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc, {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc)}sk(I),prf (ms, serverfinished, h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc), {h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc )}sk(I), ccs)));

macro clientkey=prf(ms, keyexpansionclient, ni, nr);

macro serverkey= prf(ms, keyexpansionserver, ni, nr);

protocol eaptls (I,R,CA)

{

role I

{

const SID, CerI, CerR,PMS: Data;

fresh ni: Nonce;

var nr:Nonce;

send\_1 (I, R, SID, ni);

recv\_4 (R, I, SID, nr, {CerR,pk(R)}sk(CA), req, shd);

send\_5 (I, CA, I);

recv\_6 (CA, I, {I, {CerI, pk(I)}pk(I)}sk(CA));

send\_7 (I, R, {CerI, pk(I)}sk(CA), {PMS}pk(R));

send\_8 (I,R , {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R) )}sk(I));

recv\_9 (R, I, ccs);

recv\_10 (R, I, prf(ms, serverfinished, h(m, m1)));

match (m2, prf(ms, serverfinished, h(m, m1)));

send\_11 (I, R, prf(ms, clientfinished, h(m, m1, m2)));

recv\_12 (R, I, {accesslevel}serverkey);

send\_13 (I, R, {success}clientkey);

claim\_I1 (I, Secret, CerR);

claim\_I2 (I, Secret, CerI);

claim\_I3 (I, Secret, PMS);

claim\_I4 (I, Secret, ms);

claim\_I5 (I, Secret, clientkey);

claim\_I6 (I, Secret, serverkey);

claim\_I7 (I, Nisynch);

claim\_I8 (I, Niagree);

}

role R

{

const SID, CerI, CerR, PMS: Data;

fresh nr: Nonce;

var ni:Nonce;

recv\_1 (I, R, SID, ni);

send\_2 (R, CA, R);

recv\_3 (CA, R, {R, {CerR, pk(R)}pk(R)}sk(CA));

send\_4 (R, I, SID, nr, {CerR,pk(R)}sk(CA), req, shd);

recv\_7 (I, R, {CerI, pk(I)}sk(CA), {PMS}pk(R));

recv\_8 (I,R , {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R) )}sk(I));

match(m1, {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R))}sk(I));

send\_9 (R, I, ccs);

send\_10 (R, I, prf(ms, serverfinished, h(m, m1)));

recv\_11 (I, R, prf (ms, clientfinished, h(m, m1, m2)));

match (m3, prf(ms, clientfinished, h(m, m1, m2)));

send\_12 (R, I, {accesslevel}serverkey);

recv\_13 (I, R, {success}clientkey);

claim\_R1 (R, Secret, CerR);

claim\_R2 (R, Secret, CerI);

claim\_R3 (R, Secret, PMS);

claim\_R4 (R, Secret, ms);

claim\_R5 (R, Secret, clientkey);

claim\_R6 (R, Secret, serverkey);

claim\_R7 (R, Nisynch);

claim\_R8 (R, Niagree);

}

role CA

{

const CerI, CerR: Data;

recv\_2 (R, CA, R);

send\_3 (CA, R, {R, {CerR, pk(R)}pk(R)}sk(CA));

recv\_5 (I, CA, I);

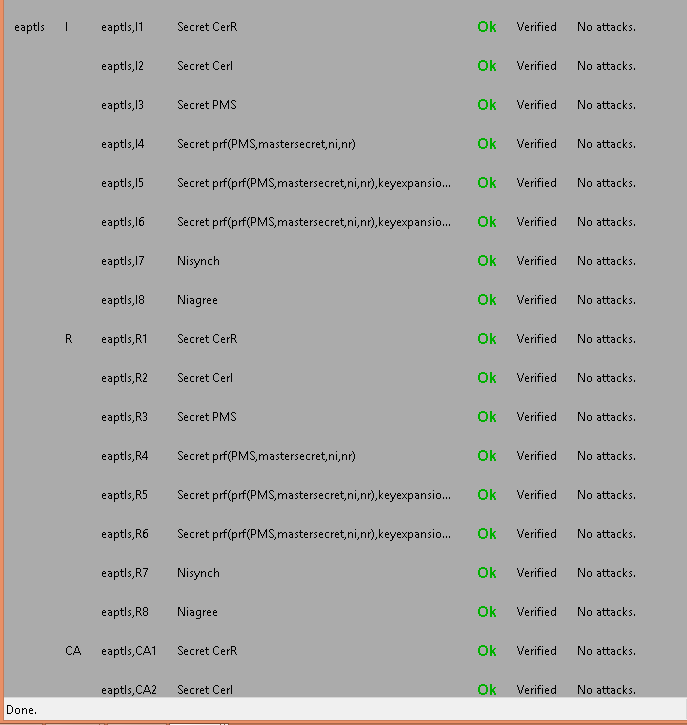
send\_6 (CA, I, {I, {CerI, pk(I)}pk(I)}sk(CA));

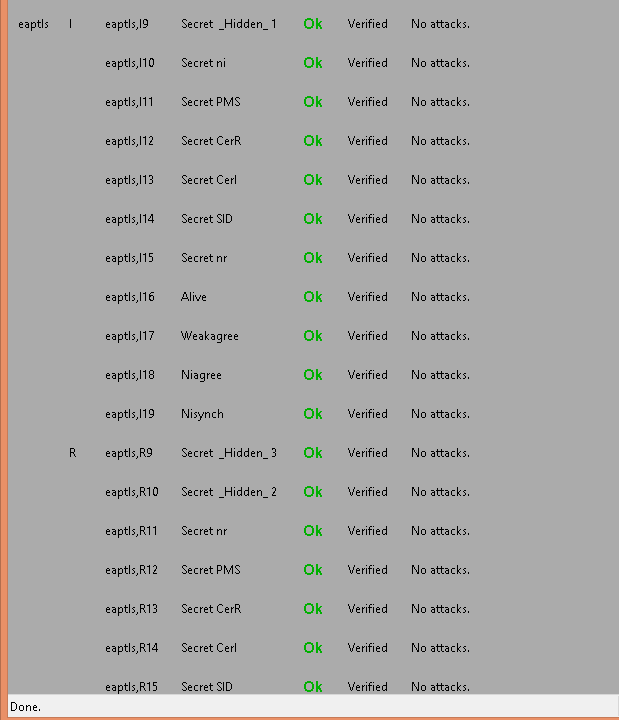
claim\_CA1 (CA, Secret, CerR);

claim\_CA2 (CA, Secret, CerI);

}

}





clc;

clear;

trsae= 0.1667 %rsaencryption

trsad= 0.1667 %rsadecryption

thmacsha256= 0.0714

tsha1= 0.0384; %sha

tsha256= 0.05 %sha256

tmd5= 0.05263; %md5

tdese= 0.29165; %desencryption

tdesd= 0.29165; %desdecryption

%EAP-TLS AUTHENTICATION

eaptlso\_a = [trsae,tsha1+trsae,5\*thmacsha256+2\*thmacsha256, 2\*thmacsha256,14\*thmacsha256+tdesd,tdese];

eaptlso\_b= [trsad,tsha1+trsae,5\*thmacsha256+2\*thmacsha256, 2\*thmacsha256,14\*thmacsha256+tdese,tdesd];

eaptlso=[trsae,trsad,tsha1+trsae,tsha1+trsad, 5\*thmacsha256+2\*thmacsha256, 5\*thmacsha256+2\*thmacsha256,2\*thmacsha256,2\*thmacsha256,14\*thmacsha256+tdese,14\*thmacsha256+tdesd+tdese,tdesd];

%total===============================

for tmp = 2:length(eaptlso\_a);

eaptlso\_a (tmp)= eaptlso\_a (tmp-1)+ eaptlso\_a (tmp);

end

for tmp = 2:length(eaptlso);

eaptlso (tmp)= eaptlso (tmp-1)+ eaptlso (tmp);

end

%EAP-TLS AUTHENTICATION Modified

eaptlsm\_a = [trsae, tsha1+trsae, 2\*thmacsha256+tsha1+thmacsha256, 4\*thmacsha256+6\*tsha1+tdesd,tdese];

eaptlsm\_b= [trsad,tsha1+trsad, 2\*thmacsha256+tsha1+thmacsha256, 4\*thmacsha256+6\*tsha1+tdese, tdesd];

eaptlsm=[trsae, trsad,tsha1+trsae, tsha1+trsad, 3\*thmacsha256+tsha256+thmacsha256,3\*thmacsha256+tsha256+thmacsha256,10\*thmacsha256+tdese,10\*thmacsha256+tdesd+tdese,tdesd];

%total===============================

for tmp = 2:length(eaptlsm\_a);

eaptlsm\_a (tmp)= eaptlsm\_a (tmp-1)+ eaptlsm\_a (tmp);

end

for tmp = 2:length(eaptlsm);

eaptlsm (tmp)= eaptlsm (tmp-1)+ eaptlsm (tmp);

end

total\_number = 100000;

unkown\_attacks= 0;

y\_eaptlso = zeros(1,10);

y\_eaptlsm= zeros(1,10);

left\_time\_eaptlso= 0;

left\_time\_eaptlsm=0;

n=1;

for x=0:0.1:0.9

left\_time\_eaptlso= total\_number\*(1-x)\*eaptlso\_a(length(eaptlso\_a));

left\_time\_eaptlsm= total\_number\*(1-x)\*eaptlsm\_a(length(eaptlsm\_a));

unknown\_attacks = uint16(total\_number\*x);

unexpected\_delay\_eaptlso = randi([1,length(eaptlso\_a)],1,unknown\_attacks);

unexpected\_delay\_eaptlsm = randi([1,length(eaptlsm\_a)],1,unknown\_attacks);

attack\_total\_delay\_eaptlso= 0;

attack\_total\_delay\_eaptlsm= 0;

for i=1:unknown\_attacks

attack\_total\_delay\_eaptlso = attack\_total\_delay\_eaptlso + eaptlso\_a(unexpected\_delay\_eaptlso(i));

attack\_total\_delay\_eaptlsm=attack\_total\_delay\_eaptlsm+eaptlsm\_a(unexpected\_delay\_eaptlsm(i));

end

y\_eaptlso(n)=(left\_time\_eaptlso+attack\_total\_delay\_eaptlso)/(total\_number\*(1-x));

y\_eaptlsm(n)=(left\_time\_eaptlsm+attack\_total\_delay\_eaptlsm)/(total\_number\*(1-x));

n=n+1;

end

x=0:0.1:0.9;

%figure;

plot(x,y\_eaptlso,'-k', x, y\_eaptlsm,'-.k');

set(gca,'XTick',0:0.1:1);

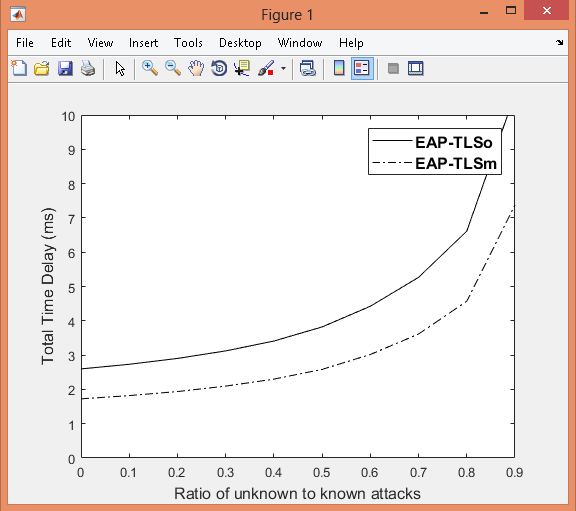
set(gca, 'xticklabel', {'0','0.1','0.2','0.3','0.4','0.5','0.6','0.7','0.8','0.9','1'});

xlabel('Ratio of unknown to known attacks’,'fontsize',12);

ylabel('Total Time Delay (ms)','fontsize',12);

legend({'EAP-TLSo', 'EAP-TLSm'},'FontSize',12,'FontWeight','bold');

axis([0,0.9,0,10]);



Significant changes:

PRF:

PRF(secret, label, seed) = P\_<hash>(secret, label + seed)

P\_hash(secret, seed) = HMAC\_hash(secret, A(1) + seed) +

HMAC\_hash(secret, A(2) + seed) +

HMAC\_hash(secret, A(3) + seed) + ...

where + indicates concatenation.

A() is defined as:

A(0) = seed

A(i) = HMAC\_hash(secret, A(i-1))

Here, older versions use md5+sha1 for calculating A(i), we will be using hmacsha256 for the same, even if not applied to EAP-TLS/EAP-TTLS/EAP-PEAP in some cases.

Here, a significant difference is obtained by starting with A(0) itself and then incrementing. (reduces one hashing in each step)

P\_hash(secret, seed) = HMAC\_hash(secret, A(0) + seed) +

HMAC\_hash(secret, A(1) + seed) +

HMAC\_hash(secret, A(2) + seed) + ...

where + indicates concatenation.

A() is defined as:

A(0) = seed

A(i) = sha1(secret, A(i-1))

key\_block = PRF(SecurityParameters.master\_secret,

"key expansion",

SecurityParameters.server\_random +

SecurityParameters.client\_random);

until enough output has been generated. Then, the key\_block is

partitioned as follows:

client\_write\_MAC\_key[SecurityParameters.mac\_key\_length]

server\_write\_MAC\_key[SecurityParameters.mac\_key\_length]

client\_write\_key[SecurityParameters.enc\_key\_length]

server\_write\_key[SecurityParameters.enc\_key\_length]

client\_write\_IV[SecurityParameters.fixed\_iv\_length]

server\_write\_IV[SecurityParameters.fixed\_iv\_length]

finished

PRF(master\_secret, finished\_label, Hash(handshake\_messages))

[0..verify\_data\_length-1];