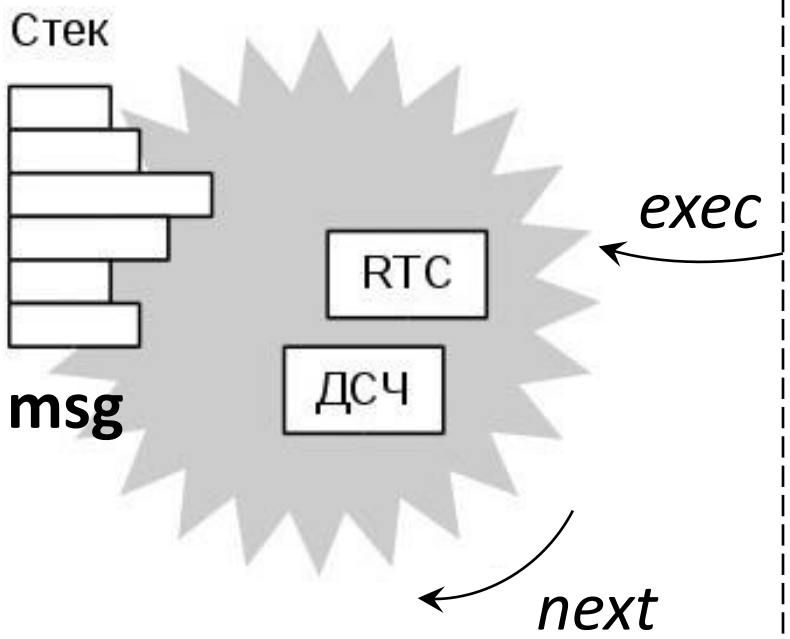
Ежегодная международная научно-практическая конференция
«РусКрипто’2020»

# Автоматическая трансляция спецификаций криптографических протоколов в нотации CMN.1 в нотацию ProVerif

Прокопьев С.Е.
ТК26

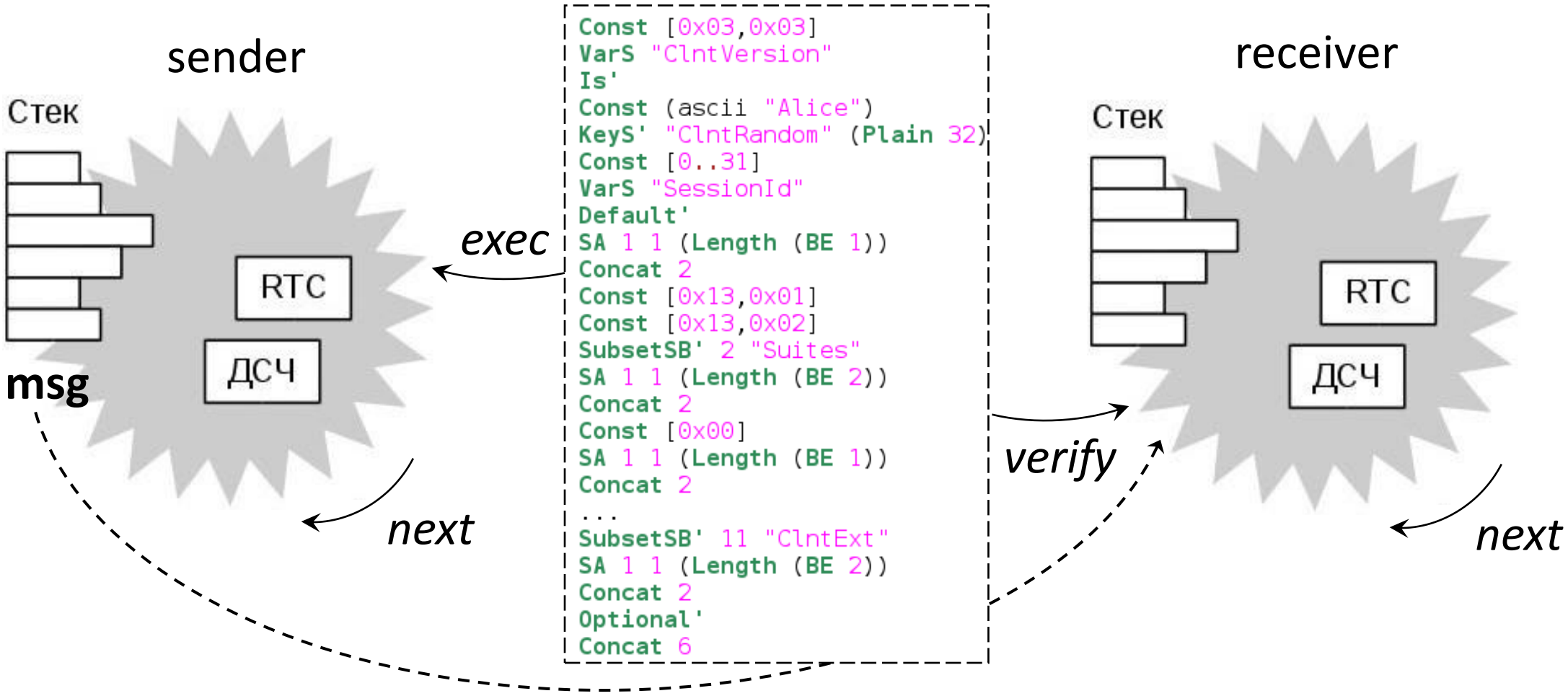
# Криптографическая стековая машина*




```
Const [0x03,0x03]
VarS "ClntVersion"
Is'
Const (ascii "Alice")
KeyS' "ClntRandom" (Plain 32)
Const [0..31]
VarS "SessionId"
Default'
SA 1 1 (Length (BE 1))
Concat 2
Const [0x13,0x01]
Const [0x13,0x02]
SubsetSB' 2 "Suites"
SA 1 1 (Length (BE 2))
Concat 2
Const [0x00]
SA 1 1 (Length (BE 1))
Concat 2
...
SubsetSB' 11 "ClntExt"
SA 1 1 (Length (BE 2))
Concat 2
Optional'
Concat 6
```

* *Prokopev S.E. Cryptographic Stack Machine Notation One. Trudy ISP RAN/Proc. ISP RAS, 2018, vol. 30, issue 3, pp. 165-182. DOI: 10.15514/ISPRAS-2018-30(3)-12.*

# Криптографическая стековая машина*




* *Prokopev S.E. Cryptographic Stack Machine Notation One. Trudy ISP RAN/Proc. ISP RAS, 2018, vol. 30, issue 3, pp. 165-182. DOI: 10.15514/ISPRAS-2018-30(3)-12.*

```
Const [0x03,0x03]
VarS "ClntVersion"
Is'
Const (ascii "Alice")
KeyS' "ClntRandom" (Plain 32)
Const [0..31]
VarS "SessionId"
Default'
SA 1 1 (Length (BE 1))
Concat 2
Const [0x13,0x01]
Const [0x13,0x02]
SubsetSB' 2 "Suites"
SA 1 1 (Length (BE 2))
Concat 2
Const [0x00]
SA 1 1 (Length (BE 1))
Concat 2
...
SubsetSB' 11 "ClntExt"
SA 1 1 (Length (BE 2))
Concat 2
Optional'
Concat 6
```

# Декларативная запись + хороший хозяин (Haskell)

```
clientHello =
 [VarS "ClntVersion" ## Const [0x03,0x03],
  KeyS "ClntRandom" (Plain 32) [name src],
  WithLen (BE 1) [VarS "SessionId" // [0..31]],
  WithLen (BE 2) [SubsetSB "Suites" ciphSuites],
  WithLen (BE 1) [Const [0x00]],
  WithLen (BE 2)
    [SubsetSB "ClntExt" extensions] # Optional]
```

# TLS 1.3 (CMN.1)

```
hshkMsg src side =
 [Var "HshkType"&src # Len 1,
  WithLen (BE 3)
   [Select (Var "HshkType"&src)
     [Case [0x01] clientHello,
      Case [0x02] serverHello,
      Case [0x05] endOfEarlyData,
      Case [0x08] encryptedExtensions,
      Case [0x0d] certificateRequest,
      Case [0x0b] certificate,
      Case [0x0f] certificateVerify,
      Case [0x14] finished,
      Case [0x04] newSessionTicket,
      Case [0x18] keyUpdate]]]


clientHello =
 [VarS "ClntVersion" ## Const [0x03,0x03],
  KeyS "ClntRandom" (Plain 32) [name src],
  WithLen (BE 1) [VarS "SessionId" // [0..31]],
  WithLen (BE 2) [SubsetSB "Suites" ciphSuites],
  WithLen (BE 1) [Const [0x00]],
  WithLen (BE 2)
    [SubsetSB "ClntExt" extensions] # Optional]
```

# TLS 1.3 ( RFC 5246, 8446)

```
struct {
  HandshakeType msg_type; /* handshake type */
  uint24 length;   /* remaining bytes in message */
  select (Handshake.msg_type) {
    case client_hello:          ClientHello;
    case server_hello:          ServerHello;
    case end_of_early_data:     EndOfEarlyData;
    case encrypted_extensions:  EncryptedExtensions;
    case certificate_request:   CertificateRequest;
    case certificate:           Certificate;
    case certificate_verify:    CertificateVerify;
    case finished:              Finished;
    case new_session_ticket:    NewSessionTicket;
    case key_update:            KeyUpdate;
  };
} Handshake;
struct {
  ProtocolVersion legacy_version = 0x0303;
  Random random;
  opaque legacy_session_id<0..32>;
  CipherSuite cipher_suites<2..2^16-2>;
  opaque legacy_compression_methods<1..2^8-1>;
  Extension extensions<8..2^16-1>;
} ClientHello;
```

# Трансляция трасс протокола в спецификации для анализаторов безопасности схем протоколов

Scyther, ProVerif, Tamarin ...

# Трансляция трасс протокола в спецификации для анализаторов безопасности схем протоколов

Scyther, ProVerif, Tamarin ...

Упрощенный вариант схемы аутентификации Нидхэма-Шрёдера*

Alice ⟶ Bob : aenc ((NonceA, pubkey (skA)), pubkey(skB))
Bob ⟶ Alice : aenc ((NonceA, NonceB), pubkey(skA))
Alice ⟶ Bob : aenc (NonceB, pubkey (skB))

** ProVerif: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.*

# Практический протокол на базе упрощенной схемы Нидхэма-Шрёдера

```
name Alice = Const (ascii "Alice")
name Bob = Const (ascii "Bob")
name TTP = Const (ascii "TrustedThirdParty")
curve = VarS "Curve" // curve192v1ID
base = VarS "BasePoint" // curve192v1Base
n = VarS "CurveN" // curve192v1N
prvkey role = KeyS "SK" role ECScalar [curve, name TTP] # Len 24
pubkey sk = ECMult [curve, base, sk] # Len 48
nonce role = KeyS "Nonce" (Nonce 16) [name role]

msg1 = (Alice, Bob, [nonce Alice, pubkey (prvkey Alice)])
msg2 = (Bob, Alice, [nonce Alice, nonce Bob])
msg3 = (Alice, Bob, [nonce Bob])
msgPrvkey r = [prvkey r]

encrypted (src, dst, plaintext) =
   [WithLen (BE 2) [r], WithLen (BE 2) [s], ivector,
    Encrypt ctxCBC_AES192 [B plaintext, ivector, symmkey]] where
        k = Key "EphemK" ECScalar [curve, name src]
        r = ECMult [curve, base, k]
        s = ModMult [symmkey, FirstHalf dhkey, n] where
           dhkey = ECMult [curve, ECMult [curve, base, prvkey dst], k]
        ivector = Key "IVec" (Plain 16) [name src]
        symmkey = Key "SymmK" (Plain 24) [name src]
```

```
main = do
  roles [Alice, Bob, TTP]
  roleNames
    [(Alice, name Alice),
     (Bob, name Bob),
     (TTP, name TTP)]
  messagePriv 1 TTP Alice (prvkeyMsg Alice)
  messagePriv 1 TTP Bob (prvkeyMsg Bob)
  publish TTP (pubkey (prvkey Alice))
  message TTP Alice (pubkeyMsg Bob)
  message Alice Bob (encrypted plainMsg1)
  message Bob Alice (encrypted plainMsg2)
  message Alice Bob (encrypted plainMsg3)
  messagePriv 2 TTP Alice (pubkeyMsg Bob)
  messagePriv 2 TTP Bob (pubkeyMsg Alice)
```

# Фрагмент сгенерированной ProVerif-спецификации

```
let processAlice() =
  in(dTTPAlice1,kXAlice_ECScalar5:bitstring);
  in(c,f9:bitstring);
  new kN_Nonce24:bitstring;
  new kIVec4_Plain23:bitstring;
  new kSymmK4_Plain14:bitstring;
  new kEphemK4_ECScalar10:bitstring;
  let vCurve = curve192v1ID in
  let vBasePoint = curve192v1Base in
  let f25 = ECMult(vCurve,vBasePoint,kXAlice_ECScalar5) in
  let vCurveN = curve192v1N in
  let f20 = ModMult(kSymmK4_Plain14,mFirstHalf(ECMult(vCurve,f9,kEphemK4_ECScalar10)),vCurveN) in
  let f11 = ECMult(vCurve,vBasePoint,kEphemK4_ECScalar10) in
  out(c,((LengthBE2(f11),f11),(LengthBE2(f20),f20),kIVec4_Plain23,EncryptAES192
((kN_Nonce24,f25),kIVec4_Plain23,kSymmK4_Plain14)));
  in(c,((f32:bitstring,f31:bitstring),(f39:bitstring,f38:bitstring),kIVec5_Plain41:bitstring,f44:bitstring));
  if f39 = LengthBE2(f38) then
  if f32 = LengthBE2(f31) then
  let f43 = DecryptAES192(f44,kIVec5_Plain41,ModDiv0(f38,mFirstHalf(ECMult(vCurve,f31,kXAlice_ECScalar5)),vCurveN)) in
  let (=kN_Nonce24,kN_Nonce42:bitstring) = f43 in
  new kIVec6_Plain61:bitstring;
  new kSymmK6_Plain54:bitstring;
  new kEphemK6_ECScalar50:bitstring;
  let f58 = ModMult(kSymmK6_Plain54,mFirstHalf(ECMult(vCurve,f9,kEphemK6_ECScalar50)),vCurveN) in
  let f51 = ECMult(vCurve,vBasePoint,kEphemK6_ECScalar50) in
  out(c,((LengthBE2(f51),f51),(LengthBE2(f58),f58),kIVec6_Plain61,EncryptAES192
(kN_Nonce42,kIVec6_Plain61,kSymmK6_Plain54)));
  in(dTTPAlice2,=f9);
```

# Моделирование асимметричного шифрования

forall m : bitstring, y : skey;  adec (aenc (m, pk (y)), y) = m.

# Моделирование асимметричного шифрования

```
forall m : bitstring, y : skey;  adec (aenc (m, pk (y)), y) = m.
```

---

```
forall a : bitstring,  b : bitstring,  n : bitstring;
  ModDiv0 (ModMult (a, b, n), b, n) = a.
equation forall a:bitstring,b:bitstring;
  FirstHalf (ECMult (curve192v1ID, ECMult (curve192v1ID, curve192v1Base, a), b)) =
  FirstHalf (ECMult (curve192v1ID, ECMult (curve192v1ID, curve192v1Base, b), a)).
```

---

# Моделирование асимметричного шифрования

```
forall m : bitstring, y : skey;  adec (aenc (m, pk (y)), y) = m.
```

---

```
forall a : bitstring,  b : bitstring,  n : bitstring;
  ModDiv0 (ModMult (a, b, n), b, n) = a.
equation forall a:bitstring,b:bitstring;
  FirstHalf (ECMult (curve192v1ID, ECMult (curve192v1ID, curve192v1Base, a), b)) =
  FirstHalf (ECMult (curve192v1ID, ECMult (curve192v1ID, curve192v1Base, b), a)).
```

---

```
forall m : bitstring, k : bitstring, y : bitstring, n : bitstring;
 ModDiv0 (ModMult (m, FirstHalf (ECMult (curve192v1ID, ECMult (curve192v1ID,
     curve192v1Base, y), k)), n), FirstHalf (ECMult (curve192v1ID, ECMult
         (curve192v1ID, curve192v1Base, k), y)), n) = m.
```