ISSN 22773061

Proof of Logic: Correctness of Next Generation Security Mechanisms

Dr. Danilo Valeros Bernardo
Db2Powerhouse Not-for-Profit Global Research Institute, Singapore, Spain, UK, USA, Philippines, Sydney Australia
bernardan@gmail.com

ABSTRACT

In this paper, three security mechanisms developed to form the UDT (UDP-Data Transfer protocol) Security Architecture are evaluated and analyzed. An approach is utilized to ascertain the applicability and secrecy properties of the selected security mechanisms when implemented with UDT. In this approach, a formal proof of correctness, through formal composition logic is carried out. This approach is modular; it has a separate proof for each protocol section that provides insight into the network environment in which each section can be reliably employed. Moreover, the proof holds for a variety of failure recovery strategies and other implementation and configuration options.

This paper is an extension and a revised version of the works published by the author.

Indexing terms/Keywords

UDT, UDT, UDT-DTLS, GSS-API, High-speed networks, PCL

Academic Discipline And Sub-Disciplines

Computer Science, Network Security, Computer Theory , Computer Logic;

SUBJECT CLASSIFICATION

Computer Logic; Computer Theory; Mathematics Subject Classification; Library of Congress Classification

TYPE (METHOD/APPROACH)

Review of Security Mechanisms developed for UDT ; Proof of Correctness of these mechanisms using Protocol Composite Logic 3. Proofs of mechanisms for the UDT Security Architecture

Council for Innovative Research
Peer Review Research Publishing System
Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY
Vol 9, No 1
editor@cirworld.com
www.cirworld.com, member.cirworld.com

INTRODUCTION

The rapidly increasing use of high-speed networks coincides with the emergence of high speed network protocols. UDT (UDP-based data transfer protocol) was developed in 2007 [35] to complement TCP's existing constraints when TCP is used in transferring bulk data across long distances. However, UDT, a fairly new protocol, was designed in absence of thorough security compositional features. In this work, we explore various security mechanisms available and demonstrate their applicability to secure UDT data transmissions.

Experiments and simulations were conducted [4-15] on these mechanisms and noted that various mechanisms worked for the existing protocols — such as TCP and UDP — but did not, as expected, work for UDT. There were, however, various mechanisms found to be viable for UDT but not for UDP, such as AO and GSS-API. The combination of UDT and UDP provided UDT with the connection and flow control that it required to operate in the selected mechanisms.

The results emphasized the following:

(1) Most mechanisms presented were experimentally validated [4-15] on connection-based protocols.

(2) The use of GSS-API, HIP, and CGA were complex and costly, but provided security solutions for the new protocol. Proof of methods through formalization will be used for verification.

(3) An option such as AO is suitable for UDT, provided that an alternative or a combination of hash methods is used.

(4) Public Key can be used, but is also costly when the application relies on certificate authorities.

(5) The results provide additional opportunity to address security for UDT and other protocols. They can assist network and security investigators, designers, and users, all of whom consider and incorporate security when implementing UDT across wide area networks. These can also support the security architectural designs of UDP-based protocols, as well as assist in the future development of other state-of-the-art fast data transfer protocols.

The approach is outlined in the next sections of this paper by conducting rigorous theoretic proof of correctness. It focuses on three mechanisms found provable within UDT implementations. By provable, we mean that these mechanisms were successfully simulated and tested in UDT practical environments, albeit with minor changes required.

To analyze the protocol based on formal language and logic, we employ the PCL (Protocol Composite Logic) method developed in the Security Lab at Stanford.

PCL [24] entails reasoning about properties achieved by formalized steps in a setting that does not compel explicit reasoning about attacker actions. Many literatures define PCL as a formal approach for proving security properties of a class of network protocols. According to [1,2,22,24-27,30-31,36,38-39], the central question addressed by PCL is whether it is possible to prove properties of security protocols compositionally, by using reasoning steps that do not explicitly mention attacker actions.

In order to reason about the protocols, the proof of properties of one sequence of actions by one agent involves not only local reasoning about the security goal of that component [37-39], but also about environmental conditions that prevent destructive interference from other actors that may use the same certificates of key materials, according to [38].

These environmental conditions are generally formulated as protocol invariants. These are properties that are true for all of the roles of the protocol at hand, and according to [38-40], there are properties that may be required in any other protocol operating in the same environment.

In this paper, proving the correctness of each selected security mechanisms for UDT (UDP-Transport Protocol) in the symbolic model [30], and determining any issues in its implementation through employing formal logic highlight important characteristics of these mechanisms . Connections between symbolic trace properties and computational soundness properties are achieved in [37]. All these efforts have been directed at proving security properties for well-established mechanisms.

CONTRIBUTION

The major contribution of this paper is in its rigorous work aiming to achieve proof of correctness of the proposed UDT Security Mechanisms through Protocol Composite Logic.

Method

We begin with a brief discussion of PCL relevant to the analysis, [1,2,26]. We base our discussion on logic notations introduced by the original proponents of PCL [26], which we fully acknowledge in this paper. As of this writing, new PCL notations have been created. However, just like other mathematical notations that already exist, we use the current logic notations for our technique to analyze our newly created security mechanisms for UDT since these notations are mature and successfully used to analyze existing mature protocols.

In this paper, we outline the proof system and the proof of soundness of the axioms [26] and the rules [1,2,22,24-27,30-32,36-39]. Most protocol proofs employ formulas of the form $\theta[P]X\varphi$, which expresses that initiating from a state where formula θ is true, after actions P are determined and executed by the thread X, the formula φ is true in the resulting state.

Formulas φ and θ typically create assertions about temporal order of actions and the data accessible to various principals that are useful for stating secrecy [38-39].

PROOF OF UDT-AO PROTOCOL

The first mechanism we propose is UDT-AO (Authentication Option) protocol. It is a lightweight protocol part of our ongoing IETF review process for UDT. We show how UDT-AO is intended to secure long-lived connections for UDT when used in various routing protocols. It is not intended to replace IPsec suite to secure connections. Hence, we analyze UDT-AO protocol for consideration in the development of a viable security architecture. We employ a finite-state method to ascertain that this protocol does not have any flaws. We also substantiate the protocol utilising a protocol verification logic. We use formal proof to verify the viability of this protocol to secure UDT transmission.

UDT is a connection-oriented protocol. As such, it requires to include an OPTION for authentication when it is used in data transmission. This is because its connections, like TCP, are likely to be spoofed [43].

The proposed option can be implemented on Type 2 of the UDT header. This field is reserved to determine specific control packets in the Composable UDT framework. Every segment sent on a UDT connection to be secured against spoofing will similarly contain the 16-byte MD5 digest achieved by applying the MD5 algorithm.

The UDT packet header and UDP pseudo-header are in network byte order. The nature of the key is deliberately left unspecified, but it must be known by both ends of the connection, similar with TCP [16,17-18,29,44]. However, a particular UDT implementation will determine what the application may specify as the key.

The focus is on validating the protocols and their applicability to UDT by determining if errors and incompatibility problems exist, and, therefore, in their absence re-enforce the viability of AO for UDT security architecture.

UDT-AO provides message authentication verification between two end points [4-15]. This message authentication function protects a message’s data integrity [15]. In order to accomplish this function, Message Authentication Codes (MAC) are utilised, which rely on Shared Keys (SK). There are various ways to generate MACs. The general requirements are outlined for MACs used in UDT-AO, both for currently specified MACs and for any future specified MACs. Two MACs algorithms selected that are necessary in all UDT-AO implementations. Moreover, two Key Derivation Functions (KDFs) employed to create traffic keys used by the MACs are introduced. These KDFs are required by all UDT-AO implementations, as presented (Table 1) below.

Table 1. Successful Message Exchange in UDT-AO

[Message 1:S . P]: SNonce, S, AlgocryptList
[Message 2:P . S]: P, S, PNonce, SNonce, AlgocryptList, AlgocryptSel,
{Payload}KDF(PKEY),MACSK
[Message 3:S . P]: PNonce, SNonce, AlgocryptSel, {Payload}KDF
(PKEY),MACSK
[Message 4: P . S]:{Payload}KDF (PKEY),MACSK

Successful UDT-AO message transfer exchange.

The Keys Shared (SKey), (PKey), and Private Keys (PSK) are derived from a key derivation function KDF, which names a Pseudorandom Function (PRF) and uses a Master_Key (MKey) and some connection-specific input with that PRF to generate Traffic_Keys (TKey), the keys suitable for authenticating and integrity-checking individual UDT segments.

UDT-AO Description

MKey is generated as seed for the KDF. It is similarly assumed this is a readable PSK; thus, it is also considered, which based on the characteristics of existing protocols, it is of variable length. MKey should be random, but in some cases when chosen by the user, it might not be. For interoperability, the management interface by which the PSK is configured [40] must acknowledge ASCII strings, and must also permit for configuration of any arbitrary binary string in hexadecimal form.

The assumption is that KDF-X selects two arguments, a key and a seed, and outputs a bit string of length X [2]. The notation KDF-X(Y,Z)[i..j] constitutes the *i’th* through *j’th* octets (8 bits) of the output of the KDF-X. The PSK has length PL, while the SKey and PKey have length KS, which is a value specified by the Algocrypt.

Similarly, the first 128 octets of KDF-{128+2*KS}(MKey, inputString)[128+KS..127+2*KS] are divided into two keys [40] which are exported as part of the protocol. They may be employed for key derivation in higher level protocols [1,2]. Every AO method which supports key derivation is needed to export such keys, but they have been omitted because they are not relevant to the current analysis.

UDT-AO is designed to equip mutual basic authentication between the peer and the server (end-to-end). The successful message exchange decides the authenticity of the peer by the use of key SKey, which is deduced from the long-term key

ISSN 22773061

PSK for MAC in Message 2 in the algorithm. The successful message exchange purports [40] the authenticity of the server by the use of SKey for the MAC in Message 3. AO is also designed to cater for session independence. This means that even if there is a weakened exchange, this prevents the attacker from compromising past or future sessions. AO is a symmetric key authentication protocol, and therefore the secrecy of long-term key PSK is essential for all the above introduced properties to hold [24-26].

UDT-AO PROOF OF CORRECTNESS

In this section, the introduction of a formal correctness proof of UDT-AO using a formal language method that executes any number of principals and sessions, over both symbolic models and over more traditional cryptographic assumptions is presented.

UDT-AO Security Properties

The properties that UDT-AO ought to satisfy include:

Setup Assumption. To establish security properties of the UDT-AO protocol, [1,2,24-27], it is deduced that the Server$\hat{S}$ and the Peer$\hat{P}$ in consideration are both honest, and are the only parties which recognise the corresponding shared *PSK*. However, this acquiesces all other principals in the network to be potentially malicious and capable of reading, blocking and changing messages transmitted to the network.

Definition 1 (Secrecy). *The Server to Peer is said to exchange key secrecy, where defined as:*

$\varphi setup \equiv Honest(\hat{P}) \wedge Honest(\hat{S}) \wedge (Has(X, PSK) \supset \hat{X} = \hat{S} \vee \hat{X} = \hat{P})$

Security Theorems. The secrecy theorem for UDT-AO inculcates that the signing and encryption keys SKey and PKey should not be obvious and known to any principal other than the peer and the server. For server$\hat{S}$ and peer$\hat{P}$, this property is used as *SECudt-ao*(*S,P*) defined as:

$SECudt\text{-}ao(S, P) \equiv (Has(X, PKey) \vee Has(X, SKey)) \vee (\hat{X} = \hat{S} \supset \hat{X} = \hat{P})$

Theorem 1 (AO-Secrecy). *On execution of the server role, key secrecy holds. Similarly for the peer role. Formally,* UDT-AO _ *SECserver pkey,skey , SECpeer pkey,skey, where*

SECserver pkey,skey≡ [**UDT**: **Server**]*S SECudt-ao*(*S, P*)

SECpeer pkey,skey≡ [**UDT**: **Peer**]*P SECudt-ao*(*S, P*)

Proof Sketch. *Proof intuition is as follows:*

PSK is considered to be known to $\hat{P}$ and $\hat{S}$ only. The keys SKey,PKey are determined by employing PSK in a key derivation function (MKEY could be a truncation of PSK or generated by application of a PRG to PSK, according to the length needed). The honest parties employ SKey,PKey as only encryption or signature keys - none of the payloads are determined by a KDF application. This is the intuition why SKey,PKey remain secrets. A rigorous proof would utilize a stronger induction hypothesis and induction over all honest party actions[26].

The authentication theorem for UDT-AO creates that on completion of the protocol, the principals accede on each other's identity, protocol completion status, the cryptographic suite list and selection, and each other's nonces. The authentication property for UDT-AO is determined in terms of matching conversations[2]. The basic idea of matching conversations is that on execution of a server role, we corroborate that there exists a role of the designated peer with a corresponding view of the interaction [40].

For server $\hat{S}$, communicating with client $\hat{P}$, matching conversations are created as *AUTHudt-ao*(*S, P*) defined below:

$$
\begin{aligned}
AUTHudt\text{-}ao(S, P) \equiv\ & (Send(S, msg1) < Receive(P, msg1)) \wedge \\
& (Receive(P, msg1) < Send(P, msg2)) \wedge \\
& (Send(P, msg2) < Receive(S, msg2)) \wedge \\
& (Receive(S, msg2) < Send(S, msg3))
\end{aligned}
$$

Definition 2 *Server is said to execute and formulate authentication session for the authenticator.*

Theorem 2 (AO-Authentication). *On formulation of the server role, authentication holds. Similarly for the peer role. Formally,* UDT-AO _ *AUTHserver peer ,AUTHpeer server, where*

$AUTHserverpeer \equiv [UDT: Server]\ S\ \exists \eta.\ P = (\hat{P}, \eta) \wedge AUTHudt\text{-}ao(S, P)$

$AUTHpeer\ server \equiv [UDT: Peer]\ P\ \exists \eta.\ S = (\hat{S}, \eta) \wedge AUTHudt\text{-}ao(S, P)$

Proof Sketch. We formulate the proof intuition here. We required to add two new axioms **MAC0** and **VMAC** (see Section 5.2.5) to the extant PCL proof system in order to contend about MACS. Axiom **MAC0** says that anybody calculating a *mac* on a message *m* with key *k* must include both *m* and *k*. Axiom **VMAC** says that if a *mac* is proven to be correct, it must have been generated by a *mac* action.

AUTHserver peer: The Server validates the *mac*1 on *msg*2 to be a mac with the key *SK*. By axiom **VMAC**, it must have been formulated by a *mac* action and by **MAC0**, it must be by someone who has *SKey*. Hence by secrecy, it is either *P* or *S* and therefore, in either case, an honest party. It is an invariant of the protocol that a *mac* action on a message of the form *Xˆ.Y.XNonce.Y*

Nonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc is executed by a thread of ˆX, captured by Γ1 - hence it must be a thread of ˆ*P*, say *P*. Also using *Γ*1 it ascertain that *P* received the first message and generated nonce *PNonce* and sent it out first in the message *msg*2. From the actions of *S*, its newly determined *SNonce* is sent out first in *msg*1. Employing this information and axioms **FS1***,* **FS2**, the sequence order then actions as receive and send as described in *AUTHserver peer*.

AUTHpeer server **:** The Peer verifies the *mac*2 on *msg*3 to be a *mac* with the key *SK*. By axiom **VMAC**, it must have been determined by a *mac* action and by **MAC0**, it must be by someone who has *SKey*. Hence by secrecy, it is some thread of either ˆ*P* or ˆ*S* and therefore, in either case, an honest party. It is an invariant of the protocol that a *mac* action on a message of the form *YNonce.XNonce.ˆY.ALGOCRYPTLIST.enc*, is performed by a thread of ˆ*Y* , captured by *Γ*2 - hence it must be a thread of ˆ*S*, say *S*.

However this *mac* does not restrict the variables *ALGOCRYPTSEL* and *enc*1 sent in *msg*2. So to ascertain that *S* received the exact same message that *P* sent, we utilise *Γ*2 to further reason that *S* verified a *mac* on a message of the form of *msg*2. And axioms **VMAC***,* **MAC0** repeat to deduce that this *mac* [40] was generated by threads of ˆ*S* or ˆ*P*. Now, it is possible to use *Γ*1 and the form of *msg*2 to contend that a thread of ˆ*P* did it, which also generated *PNonce* - hence by **AN1**, it must be *P* itself. Now an invariant can be managed that states that a thread initiating such a *mac* does it uniquely, captured by *Γ*3, thus binding *ALGOCRYPTSEL, enc*1. Moreover, **FS1***,* **FS2** can now be managed as in the previous proof to create the order described in *AUTHpeer server*.

UDT-AO Axioms

The proof system enhances first-order logic with axioms and proof rules for protocol actions, temporal reasoning, properties of security (e.g., cryptographic) primitives, and a specialised form of program invariance rule called *honesty rule* [26-27]*.* Below is the list of axioms employed in this paper.

Table 2. *New Axioms*

Formal Proofs

New Axioms

MAC0 Mac(X, m, k) ⊃Has(X,m) ^ Has(X, k) *means anybody computing a mac on a message m with k must possess both m and k.*

VMAC VerifyMac(X, m',m, k) ⊃∃Y. Mac(Y,m, k) ^m' = MAC[k](m) *states that if a mac is verified to be correct, it mush have been generated by a mac action*

Note: Extant PCL proof system reason about MACS through the new axioms MAC0 and VMAC

Invariants

Γ1 ≡ Mac(Z, ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc,K) ⊃ ˆ Z = ˆX ^ (Receive(Z, Y Nonce. ˆY .ALGOCRYPTLIST) - *assign Γ1 invariant for AUTHserver peer. – it captures the mac action on a message of the form XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc, which is performed by a thread X.ˆ, captured by Γ1*

< Send(Z, ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc.mac)) ^

mac = MAC[K](ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc) ^

FirstSend(Z,XNonce, ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc.mac)

Γ2 ≡ Mac(Z, Y Nonce.XNonce. ˆY .ALGOCRYPTLIST.enc,SKEY) ^SKEY = KDF1[K](YNonce. ˆY .XNonce.ˆX)⊃

ˆZ = ˆ Y ^ ∃ ALGOCRYPTSEL', enc1. (Send(Z, Y Nonce. ˆY .ALGOCRYPTLIST) <

Receive(Z, Yˆ .Xˆ.Y Nonce.XNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1.mac1) <

Send(Z, Y Nonce.XNonce.ALGOCRYPTLIST.enc.mac)) ^

mac1 = MAC[SKEY](Yˆ .Xˆ.Y Nonce.XNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1) ^

mac = MAC[SKEY](Y Nonce.XNonce.ALGOCRYPTLIST.enc) ^

VerifyMac(Z,mac1, Yˆ .Xˆ.Y Nonce.XNonce.ALGOCRYPTLIST.ALGOCRYPTSEL' .enc1, SKEY) ^

FirstSend(Z, Y Nonce, Y Nonce. ˆY .ALGOCRYPTLIST) - *assign Γ2 invariant for AUTHpeerserver*

Γ3 ≡ Mac(Z, ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc,K) ^

Mac(Z, ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc' ,K) ⊃ ALGOCRYPTSEL =

ALGOCRYPTSEL' ^ enc = enc' *assign Γ3 invariant for AUTHpeerserver*

Formal Proof of AUTHserver

peer

AA1 [**UDT-AO : Server**]S VerifyMac(S, mac1, ˆ P. ˆ S.PNonce.SNonce. (1)

ALGOCRYPTLIST.ALGOCRYPTSEL.enc1,SKEY)

Axiom VMAC is generated by a mac action and by MAC0, it be someone with SKEY

SECserver **VMAC** [**UDT-AO : Server**]S ∃X. (ˆX = ˆ P ∨ˆX = ˆ S) ^ (2)

pkey,SKey , Mac(X, P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc1,SKEY)

AUTHserver peer verifies the mac1 on msg2 to be a mac with key SKEY

Γ1 [**UDT-AO : Server**]S ∃η. P0 = (ˆ P, η) ^

Mac(P0, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc1,SKEY) ^

By using Γ1 we prove that P received the first message and generated nonce PNonce and sent it out first in the message msg2.

Receive(P0,msg1) < Send(P0,msg2) ^

FirstSend(P0, PNonce,msg2) (3)

(3) temporary predicate requires only until the same nonce used by the peer succeeds in completion

InstP0→P[**UDT-AO:Server**]SMac(P, ˆP.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc1,SKEY) ^

Receive(P,msg1) < Send(P,msg2) ^

FirstSend(P, PNonce,msg2) (4)

(4) temporary predicate requires only until the same nonce used by the peer succeeds in completion

FS1 [**UDT-AO : Server**]S FirstSend(S, SNonce,msg1)

order the receives and sends (5)

FS2, [**UDT-AO : Server**]S (Send(S,msg1) < Receive(P,msg1)) ^ *order the receives and sends*

(Receive(P,msg1) < Send(P,msg2)) ^

(Send(P,msg2) < Receive(S,msg2)) (6)

AA4 [**UDT-AO : Server**]S (Receive(S,msg2) < Send(S,msg3)) (7)

AUTHserver (8)

peer

Formal Proof of AUTHpeerserver

AA1 [**UDT-AO : Peer**]P VerifyMac(P,mac2, PNonce.SNonce. ˆ S.ALGOCRYPTLIST.enc2, SKEY) (9)

SECserver **VMAC** [UDT-AO : Peer]P ∃X. (ˆX = ˆ P ∨ˆX = ˆS) ^

pkey,SKey , Mac(X, PNonce.SNonce. ˆ S.ALGOCRYPTLIST.enc2, SKEY) (10)

Γ2, [**UDT-AO : Peer**]P ∃η. S0 = (ˆ S, η) ^

∃ALGOCRYPTSEL', enc1'. (Send(S0, SNonce. ˆ S.ALGOCRYPTLIST) <

Receive(S0, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1'.mac1) <

Send(S0,PNonce.SNonce.ALGOCRYPTLIST.enc2.mac)) ^

mac1 = MAC[SKEY](ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1') ^

mac = MAC[SKEY](PNonce.SNonce.ALGOCRYPTLIST.ˆS.enc2) ^

FirstSend(S0, SNonce, SNonce. ˆ S.ALGOCRYPTLIST) (11)

Inst S0 → S [**UDT-AO : Peer**]P ∃ALGOCRYPTSEL', enc1'. (Send(S, SNonce. ˆ S.ALGOCRYPTLIST) <

Receive(S, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1'.mac1) <

Send(S, PNonce.SNonce. ˆ S.ALGOCRYPTLIST.enc2.mac)) ^

mac1 =MAC[SKEY](ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1') ^

mac = MAC[SKEY](PNonce.SNonce. ˆ S.ALGOCRYPTLIST.enc2) ^

VerifyMac(S,mac1, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1', SKEY) ^

FirstSend(S, SNonce, SNonce. ˆ S.ALGOCRYPTLIST) (12)

Inst ALGOCRYPTSEL', enc1', [**UDT-AO : Peer**]P ∃X. (ˆX = ˆ P ∨ˆX = ˆS) ^

VMAC,MAC0 Mac(X, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1',SKEY) (13)

Γ1,**AA1**, [**UDT-AO : Peer**]P New(X, PNonce) ^ New(P, PNonce) (14)

AN1, [**UDT-AO : Peer**]P X = P *AN1 generated by PNonce for P thread* (15)

AA1, [**UDT-AO : Peer**]P Mac(X, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1', SKEY) ^ Mac(X, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc1,SKEY) (16)

Γ3, [**UDT-AO : Peer**]P ALGOCRYPTSEL' = ALGOCRYPTSEL ^ enc1' = enc1 (17)

[**UDT-AO : Peer**]P (Send(S,msg1) < Receive(S,msg2) < Send(S,msg3)) (18)

FS1 [**UDT-AO : Peer**]P FirstSend(P, PNonce,msg2) *order receives and sends* (19)

FS2, [UDT-AO : Peer]P (Send(S,msg1) < Receive(P,msg1)) ^ *order receives and sends*

(Send(P,msg2) < Receive(S,msg2)) ^

(Receive(S,msg2) < Send(S,msg3)) (20)

AA4 [UDT-AO : Peer]P (Receive(P,msg1) < Send(P,msg2)) (21)

AUTHpeer (22)

Server

Axioms define general truths applicable to every protocol [26-27]. For instance, the axiom VER encodes the common property of signatures [1,2,26] that if a thread verifies that a message *x* is assigned by a principal *Y^*, it must be *Y^* signature key used to generate the signature. Further, if the agent Y^ is honest, no one else has access to this key, implying that there exists a thread of the agent *Y* that did indeed sign the term *x*, according to [24-27].

UDT-AO Operating Environment

The formal proof outlined in the preceding section applies to the case where fresh nonces are generated every time. When the peer employs the same nonce repeatedly until it succeeds in completion [40], a different form of reasoning needs to be utilised to ascertain the intended message ordering. Specifically, the predicate FirstSend(*P, PNonce,msg*2) does not necessarily hold anymore. However, it is possible to still appeal to the fact that a MAC must have been generated and distributed before it could be received and verified, to be able to sequentially order messages. Therefore, formalizing this requires the new axiom *VMAC*:

VMAC_ Receive(X,m2) ^ Contains(m2,m') ^VerifyMac(X,m',m, k) ^

Mac(X,m, k)⊃ ∃Y,m1. Mac(Y,m, k) ^ Contains(m1,m') ^

(Send(Y,m1) < Receive(X,m2))

The proof above uses axioms previously proved sound in the symbolic model.

PROOF OF UDT+DTLS PROTOCOL

In this section, we outline the DTLS with UDT protocol. In UDT+DTLS, we focus on two principals called the UDT+DTLS client and the UDT+DTLS server. In a way correlative to TLS, DTLS guarantees mutual authentication and establishes a shared key between these two principals. The proof of UDT+DTLS, therefore, lies on the authentication property. The identification of any UDT+DTLS program invariants also emphasizes the security properties of UDT+DTLS as part of the development of the security architecture.

UDT-DTLS Description

We outline DTLS protocol in the formal language we introduced in the earlier sections. DTLS protocol provides end-to-end security; it is selectively deployed on the Internet in some security and e-commerce systems. We focus on how DTLS can be used to mutually authenticate the supplicant and the authenticator, and to derive a shared secret key [1,2,24-26,30] to add security in UDT data transmissions. We will be proving DTLS in isolation and will be identifying conditions under which other protocols may operate concurrently without introducing any vulnerabilities. Identifying such conditions appears valuable, given the promising deployment of DTLS on UDT. We employ the terms client and server for DTLS protocol participants, and similarly, we adhere to the proof of correctness of DTLS based on TLS, when deploying UDT.

UDT-DTLS Proof of Correctness

DTLS has many possible modes of operation. Similarly, we limit our attention to the mode where both the server and the client have certificates, since this mode satisfies the mutual authentication property. DTLS is developed to construct over datagram to cater for unreliable packet transmission, retransmission and reordering. To the greatest extent, DTLS is identical to TLS, however unlike TLS, DTLS adds explicit state to records and adds explicit sequence numbers to secure datagrams.

UDT-DTLS Security Properties

The properties that DTLS (based on TLS) ought to satisfy include:

1. Like TLS, the DTLS principals accede on each other's identity [42] protocol completion status, the values of the protocol version, cryptographic suite, and the secret that the client sends to the server. For server ˆY communicating with client ˆX, this property is formulated in Definition 3.
2. The secret that the client formulates should not be known to any principal other than the client and the server [1,2,4-15,26,42]. For server ˆY and client ˆX , this property is generated in Definition 4.

Definition 3. (DTLS Authentication, similar to TLS [28]**)**

DTLS is said to formulate session authentication for the server role if Dtls,auth holds, where

Dts,auth ::= Honest(ˆX) ^ Honest(ˆ Y) ⊃ ∃X.ActionsInOrder(

Send(X, ˆX , ˆY ,m1),

Receive(Y, ˆX, ˆY ,m1),

Send(Y, ˆ Y , ˆX ,m2),

Receive(X, ˆ Y , ˆX ,m2),

Send(X, ˆX , ˆY ,m3),

Receive(Y, ˆX, ˆY ,m3),

Send(Y, ˆ Y , ˆX ,m4))

and m1, m2, m3, m4 represent the corresponding DTLS messages.

Definition 4 (DTLS Key Secrecy). DTLS is said to provide secrecy if Dtls,sec holds, where

Dtls,sec ::= Honest(ˆX) ^ Honest(ˆ Y) ⊃

Has(ˆX , secret) ^

Has(ˆ Y , secret) ^

(Has(ˆ Z, secret) ⊃ ˆ Z = ˆX v ˆ Z = ˆ Y)

The proof system is used to prove guarantees for both the client and the server. Due to space constraints, the list only includes the guarantee for the authenticator in Theorem 3. The client guarantee is similar. The secrecy of the exchanged key material in TLS is established by combining local reasoning based on the client's actions with global reasoning about actions of honest agents. Intuitively, a client that generates the secret only sends it out either encrypted with an honest party's public key or uses it as a key for a keyed hash (this is captured by the predicate NonceSource). Furthermore, no honest user will ever decrypt the secret and send it in the clear. Specifically, an honest party can send the secret in the clear only if it receives it in the clear first. Secrecy follows directly from these two facts.

Theorem 3 (DTLS Server Guarantee).

(1) On execution of the server role, key secrecy and session authentication are guaranteed if the formulas in (2) hold.

Formally,

Dtls,1 ^ Dtls,,2 |-DTLS:Server]X Dtls,auth ^ Dtls,,sec

UDT-DTLS Operating Environment

We now characterize the class of protocols that safely constitutes with DTLS. As in the preceding section, we relate DTLS invariants to deployment considerations.

DTLS,1 states that messages of a certain format should not be sent out by any protocol that executes in the same environment as TLS. One set of terms exhibit keyed hashes of the handshake, where the key is the shared secret established by a DTLS session; [4-15,26,28] another set refers to signatures on the handshake messages. A client running a protocol that signs messages indiscriminately could instigate the loss of the authentication property. Such an attack would only be possible if the client certificate used by DTLS was shared with other protocols and was infringed by them.

PROOF OF UDT+GSS-API (KERBEROS) PROTOCOL

The third mechanism proposed is UDT+GSS-API. We illustrate the proof system in this section using GSS-API and we focus on Kerberos V5 [19-20,23], proven to any protocols, which GSS-API uses. In this section we illustrate how Kerberos is formalized to achieve proofs of secrecy and authentication.

UDT+GSS-API (Kerberos) Description

The formulation is based on the A level formalization of Kerberos V5. Kerberos provides mutual authentication and establishes keys between Clients and application Servers, employing a sequence of two message interactions with trusted parties called the Kerberos Authentication Server (KAS), and the Ticket Granting Server (TGS) [21,35,37].

Proof of UDT+GSS-API through Kerberos

Mechanisms are denoted in a process calculus by defining a set of roles [26], such as 'Client', or 'Server.' Each role is provided by a sequence of actions such as sending or receiving a message, generating a new nonce, or decrypting or encrypting a message. In a run of a mechanism, a principal may execute one or more instances of each role, each execution constituting a thread identified by a pair (^X;η), where ^X is a principal and η is a unique session identifier. Kerberos has four roles[36]: Client, KAS, TGS and Server. The pre-shared long-term keys between the client and KAS, the KAS and TGS, and the TGS and the application server, will be written as $k_{X;Y}^{type}$ where X and Y are the principals sharing the key. The type appearing in the superscript indicates the relationship between X and Y: c$\rightarrow$k indicates that X is acting as a Client and Y is acting as a KAS, t$\rightarrow$k for TGS and KAS and s$\rightarrow$t for application server and TGS.

In the first stage, the Client (C) generates a nonce (represented by new n1) and sends it to the KAS (K) along with the identities of the TGS (T) and itself. The KAS generates a new nonce (AKey - Authentication Key) [36,38] to be utilised as a session key between the Client and the TGS. It then sends this key along with some other fields to the client encrypted under two different keys- one it shares with the Client ($k^{c\rightarrow k}_{C,K}$) and one it shares with the TGS($k^{t\rightarrow k}_{T,K}$). The encryption with $k^{t\rightarrow k}_{T,K}$ is called the Ticket Granting Ticket (tgt). The Client extracts AKey by decrypting the component encrypted with $k^{c\rightarrow k}_{C,K}$ and recovering its parts using the match action which deconstructs $text_{kc}$ and associates the parts of this plaintext with AKey, η1, and T^. The ellipses (…) indicates further Client steps for interacting with KAS, TGS.

In the second stage, the Client gets a new session key (SKey - Service Key) and a service ticket (st) to converse with the application server S which takes place in the third stage. The control flow of Kerberos [36] exhibits a staged architecture where once one stage has been completed successfully, the subsequent stages can be performed multiple times, or aborted and started over for handling errors.

GSS-API Kerberos Properties and Operating Environment

The security objectives of proving Kerberos are of two types: authentication and secrecy. The authentication objectives take the form that a message of a certain format was indeed sent by some thread of the expected principal. The secrecy objectives achieve the form that a putative secret is a good key for certain principals. For example, $AUTH^{client}_{kas}$ outlines that when C completes executing the Client role, some thread of K^ indeed sent the expected message; SEC^{client}_{akey} outlines that the authorisation key is good after execution of the Client role by C; the other security properties are related.

The proof of Kerberos Security properties clearly underscores and demonstrates an interleaving of authentication and secrecy properties, reflecting the institution behind the proposed mechanism.

CONCLUSION

In this paper, 3 mechanisms were selected and analyzed: the UDT-AO, UDT+DTLS and UDT+GSS-API, Kerberos [4-15,19-20,36] authentication protocols.

The theoretic and discussed proofs of secrecy and authentication of UDT-AO, UDT+DTLS, and UDT+GSS-API, Kerberos [4-15] demonstrate they are useful mechanisms for UDT, provided that appropriate techniques are supplemented with extensive practical validations in UDT implementations.

ACKNOWLEDGMENTS

Appreciation extended to the Security Laboratory of Stanford University and The University of Technology – Sydney . This work is an expanded and revised version of the short papers published by the author in the many literature.

REFERENCES

[1] Abadi, M., Rogaway, P. (2002), Reconciling two views of cryptography (the computational soundness of
formal encryption). Journal of Cryptology 15,103–127

[2] Adao,P. , Bana, G., Scedrov, A.(2005), Computational and information theoretic soundness and

completeness of formal encryption. CSFW18, 170-184.

[3] Bellare, M, Rogaway, PM (1994), Entity authentication and key distribution. In, Stinson, D.R. (ed.) CRYPTO 1994. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg.

[4] Bernardo, DV. and Hoang, D. (2008), “Network Security Considerations for a New Generation Protocol UDT. Presented at Proc. IEEE the 2nd ICCIST Conference, Beijing China.

[5] Bernardo, DV. and Hoang, D. (2010), “Protecting Next Generation High Speed Protecting–UDT through Generic Security Service Application Program Interface GSS-API”. Presented at 4th IEEE International Conference on Emerging Security Information, Systems and Technologies SECURWARE 2010 Venice/Mestre, Italy.

[6] Bernardo, DV. and Hoang, D. (2009), A Security Framework and its Implementation in Fast Data Transfer Next Generation Protocol UDT, Journal of Information Assurance and Security Vol 4(354-360). ISN 1554- 1010.

[7] Bernardo, DV. and Hoang, D. (2010), “A Conceptual Approach against Next Generation Security Threats: Securing a High Speed Network Protocol – UDT”, Proc. IEEE the 2nd ICFN 2010, Shanya China.

[8] Bernardo, DV. and Hoang, D. (2010), A Pragmatic Approach: Achieving Acceptable Security Mechanisms for High Speed Data Transfer Protocol-UDT SERSC. International Journal of Security and Its Applications Vol. 4, No. 4, October, 2010.

[9] Bernardo, DV. and Hoang, D. (2010), End-to-End Security Methods for UDT Data Transmissions. FGIT 2010, Korea: 383-393 LNCS, Springer,Heidelberg.

[10] Bernardo, DV. and Hoang, D. (2010), Security Analysis of the Proposed Practical Security Mechanisms for High Speed Data Transfer Protocol. AST/UCMA/ISA/ACN 2010: 100-114, Japan, LNCS Springer Verlag Germany.

[11] Bernardo, DV. and Hoang, D. (2011), “ Empirical Survey: Experimentation and Implementations of High Speed Protocol Data Transfer for Grid, 25th IEEE AINA Workshop 2011, pp. 335-340.

[12] Bernardo, DV. and Hoang, D. (2011), “Formalisation and Information- Theoretic Soundness in the Development of Security Architecture for Next Generation Network Protocol – UDT”, SECTECH Conference, Jeju Island, Korea 2011 LNCS Springer, Heidelberg.

[13] Bernardo, DV. and Hoang, D. (2011), Multi-layer Security Analysis and Experimentation of High Speed Protocol Data Transfer for GRID, International Journal of Grid and Utility Computing, in the press, October, 2011.

[14] Bernardo, DV.“UDT (2010) -Authentication Option field, An approach “ Presented at 6th IEEE International Conference of Information Assurance and Security (IAS), Atlanta, USA, August 23-25, 2010.

[15] Bernardo, DV. and Hoang, D. (2009) “Security Architecture for UDT”, Work in Progress, IETF.

[16] Bishop, S., Fairbairn, M., Norrish, P., Sewell, P., Smith, M., and Wansbrough, K., TCP, UDP, and Sockets:rigorous and ex perimentally validated behavioural specification: Volume 2: The Specification. Technical Report UCAM-CL-TR-625,Computer Laboratory, University of Cambridge, Mar. 2005. 386pp.

[17] Bonica, R., Weis,B., Viswanathan, S., Lange, A., Wheeler, O., (2007) “Authentication for TCP-based Routing and Management Protocols, “draft-bonica-tcp-auth-06, (work in progress), Feb. 2007.

[18] Brackmo, L., O’Malley, S., and Peterson, L. (1994) “TCP Vegas: New Techniques for congestion

detection and avoidance", 1994 ACM SIGCOMMConference, pages 24-25.

[19] Butler, F., Cervesato, I., Jaggard, A.D, Scedrov, A., (2006), Verifying confidentiality and authentication in kerberos 5. In, Futatsugi, K., Mizoguchi, F.,Yonezaki, N. (eds.) ISSS 2003. LNCS, vol. 3233, pp. 1–24. Springer, Heidelberg

[20] Butler, F., Cervesato, I., Jaggard, A.D, Scedrov, A., (2002), A Formal Analysis of Some Properties of Kerberos 5 UsingMSR. In, Fifteenth Computer Security FoundationsWorkshop—CSFW-15, Cape Breton, NS, Canada, pp. 175–190. IEEE Computer Society Press, Los Alamitos.

[21] CERT, 1996a. "UDP Port Denial-of-Service Attack," Advisory CA-96.01, Computer Response Team, Pittsburg, PA.

[22] Cervasato, I., Meadows, C., Pavlovic, D., (2005), An encapsulated authentication logic for reasoning about key distribution. In, CSFW-18, IEEE Computer Society, Los Alamitos.

[23] Cervesato, I., Jaggard, A. , Scedrov, A., Tsay, J.K., Walstad, C. (2005) ,Breaking and fixing publickey kerberos (Technical report).

[24] Datta,A., Derek, A., Mitchell, JC, Pavlovic, D. (2005),A derivation system and compositional logic for security protocols. Journal of Computer Security 13, 423–482.

[25] Datta,A., Derek, A., Mitchell, Warinschi, B., (2006),Computationally sound compositional logic for key exchange protocols. In, Proceedings of 19th IEEE Computer Security Foundations Workshop, pp. 321–334. IEEE, Los Alamitos 324,

[26] Datta,A., Derek, A., Mitchell,JC, Roy, A. (2007), Protocol Composition Logic (PCL). Electronic.Notes Theory. Computer. Sci. 172, 311–358.

[27] Datta,A., Derek, A., Mitchell, JC., Shmatikov, V.Turuani,M. (2005), Probabilistic polynomial time semantics for a protocol security logic. In, Caires, L., Italiano, G.F., Monteiro, L.,Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 16–29. Springer, Heidelberg

[28] Dierks, T., Rescorla, E., (2006), The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346.

[29] Duke, M., Braden, R., Eddy, W., Blanton, E., (2006): A Roadmap for Transmission Control Protocol (TCP), RFC 4614, IETF, September 2006.

[30] Durgin, N., Mitchell, JC., Pavlovic, D. (2001), A compositional logic for protocol correctness. In Proceedings of 14th IEEE Computer Security Foundations Workshop, pp. 241–255. IEEE, Los Alamitos.

[31] Durgin, N., Mitchell, JC, Pavlovic, D. (2003), A compositional logic for proving security properties of protocols. Journal of Computer Security 11, 677–721

[32] F´abrega, F., Herzog, JC., Guttman, JD. (1998),Strand spaces, Why is a security protocol correct? In, Proceedings of the 1998 IEEE Symposium on Security and Privacy, Oakland, CA, pp. 160–171. IEEE Computer Society Press, Los Alamitos.

[33] Fishbein, M., and Ajzen, I., (1975), Belief, Attitude, Intention, and Behavior: An Introduction to Theory and Research. Reading, Addison-Wesley.

[34] Floyd, S., and Fall, K. (2009) : Promoting the use of end-to-end congestion control in the Internet. IEEE/ACM Transactions on

Networking, 7(4): 458-472, 1999.

[35] Gu, Y., Grossman, R. (2007) UDT, UDP-based Data Transfer for High- Speed Wide Area Networks. Computer Networks (Elsevier

). Volume 51, Issue 7, 2007.

[36] Hasebe, L., Okada, M. (2004), Non-monotonic properties for proving correctness in a framework of compositional logic. In

, Foundations of Computer Security Workshop, pp. 97–113. [110.] Linn, J., (1996), "The Kerberos Version 5 GSS-API

Mechanism", IETF,RFC 1964, June 1996.

[37] Meadows, C.,(19 94), A model of computation for the NRL protocol analyzer. In, Proceedings of 7th IEEE Computer Security

Foundations Workshop, pp. 84–89. IEEE, Los Alamitos. [114.] Melnikov, A., Zeilenga, K., (2006)., Simple Authentication and

Security Layer (SASL) IETF, RFC 4422, June 2006.

[38] Menezes, AJ, Oorschot van , PC. and Vanstone, SA. (1997),Handbook of Applied Cryptography, CRC Press, 1997.

[39] Micciancio, D., Warinschi, B., (2004), Soundness of formal encryption in the presence of active adversaries. In, Naor, M. (ed.)

TCC 2004. LNCS,

vol. 2951, pp. 133–151. Springer, Heidelberg.

[40.] Mitchell, JC., Roy, A., Rowe, P., and Scedrov, A., 2008. Analysis of EAPGPSK authentication protocol. In Proceedings of the

6th international conference on Applied cryptography and network security (ACNS'08), Steven M. Bellovin, Angelos Keromytis,

Rosario Gennaro, and Moti Yung (Eds.). Springer-Verlag, Berlin, Heidelberg, 309-327, 2008.

[41] Neuman, C., Yu, T., Hartman, S., Raeburn, K., (1996), Kerberos Network Authentication Service (V5), IETF, RFC 1964, 1996.

[42] Rescorla, E.,Modadugu, N., (2006), “Datagram Transport Layer Security” RFC 4347, IETF, April 2006.

[43] Touch, J., (2007), "Defending TCP Against Spoofing Attacks," RFC-4953, Informational, Jul. 2007.

[44] Zhang,B., Karp, B., Floyd, S. and Peterson, L.(2003): RR-TCP: A reordering-robust TCP with DSACK. Proc. the Eleventh IEEE

International Conference on Networking Protocols (ICNP 2003), Atlanta, GA, November 2003

Author’ biography with Photo

Dr. Danilo Valeros Bernardo is an honorary researcher of a non-for profit organisation in Australasia. He holds a degree in Computer Engineering , Masters degree in Technology and Business, Applied Sciences and PhD in Computer Science.