FYI

G. Tsudik
IBM Zurich Research
02-26-96

Zurich iKP Prototype (ZIP):
Protocol Specification Document

1 Document Status

This document is provided as an FYI.

Distribution of this document is unlimited. Please send comments to the editor (see last page for the email address.) Additional information on iKP and electronic commerce in general is available via WWW at: "http://www.zurich.ibm.com/Technology/Security/"

2 Abstract

iKP - the Internet Keyed Payments protocol family - defines an architecture for secure account-based (e.g., credit card) payments over the Internet or similar open network environments. This document presents a version of iKP evolved from the one described in [2].

The main purpose of this document is to communicate technical information. Prior knowledge of iKP concepts and basic design principles is highly recommended. Readers unfamiliar with iKP are referred to the general iKP paper [1] and earlier version of the (expired) Internet Draft [2].

Furthermore, this document is concerned only with the iKP protocol particulars. Higher-layer (iKP Transaction Layer) software is discussed elsewhere [3].

3 Acknowledgements

The Zurich iKP Prototype (ZIP) represents a joint effort by Michael Steiner, Gene Tsudik and Els Van Herreweghen.

Contributions made by Ralf Hauser, Phil Janson, Steen Larsen and Michael Waidner are gratefully acknowledged.

Contents

1 Document Status **i**

2 Abstract **i**

3 Acknowledgements **i**

4 Protocol Participants **1**

5 High-Level Protocol Description **1**

5.1 Payment Authorization 2
5.2 Separate Payment Clearance/Capture 2
5.3 Refunds 3
5.4 Inquiry 3

6 Detailed Protocol Description **3**

6.1 Atomic Fields/Symbols 3
6.2 Composite Fields/Symbols 4
6.3 Typing of Message Fields 5
6.4 Opaque Fields 5
6.5 Flow Definitions 5
6.5.1 Payment with In-line Authorization Flow 5
6.5.2 Clearance Flow 6
6.5.3 Inquiry Flow 7
6.6 Protocol Options and Flags 7

7 Protocol Flow Processing **8**

7.1 Payment Authorization Protocol 8
7.1.1 INITATE Composition 8
7.1.2 INITIATE Processing and INVOICE Composition 8
7.1.3 INVOICE Processing 9
7.1.4 PAYMENT Composition 9
7.1.5 PAYMENT Processing 9
7.1.6 AUTH-REQUEST Processing 10
7.1.7 AUTH-RESPONSE Processing 10
7.1.8 CONFIRM Processing 11
7.1.9 CANCEL Processing 11
7.1.10 STATUS Processing 12
7.2 Clearance (Capture) Protocol 12
7.2.1 CLRN-REQUEST Processing 12
7.2.2 CLRN-RESPONSE Processing 13

7.3 INQUIRY Processing . . . 13

8 Fault Tolerance and Exception Handling **13**

9 Rationales and Explanations **13**

9.1 Performance . . . 14

9.2 Verification of CONFIRM and CANCEL . . . 14

9.3 Adherence to Export Rules . . . 14

9.4 Use of ENCRYPTION in iKP . . . 14

10 Editor's Address **15**

A iKP Core API **16**

A.1 iKP Core API Return/Error Codes . . . 16

A.2 Input and Output Tokens . . . 17

A.3 Protocol Option Flags . . . 17

A.4 Buyer API Functions . . . 17

A.5 Seller Functions . . . 19

A.6 Acquirer Functions . . . 20

B iKP State Diagrams **22**

4 Protocol Participants

This section (adapted from [2]) is included in order to provide a brief rehash of the iKP milieu.

The iKP protocols are designed to allow buyers to pay for goods and services (of both electronic and non-electronic nature) over the Internet while relying on existing financial clearing networks to implement the necessary payments, as suggested in the figure below.

```
+--------+                     +--------+
| Issuer |--------------------->|Acquirer|
|        |  financial networks  |        |
|   I    |<---------------------|   A    |
+--------+                     +--------+
                                 /|\  |   Internet
                                  |   |   or private
                                  |  \|/  networks
+--------+                     +--------+
|  Buyer |*********************>| Seller |
|        |      Internet        |        |
|   B    |<*********************|   S    |
+--------+                     +--------+
```

Participants in the iKP protocol are:

1. Buyers
2. Seller
3. Seller's bank
4. Buyer's bank

In the context of today's credit card systems, seller's bank is called an *acquirer* because it acquires paper charge slips from sellers. Buyer's bank is called an issuer because it issues charge cards to buyers. In the context of iKP, an acquirer functions as a gateway between the Internet and existing financial networks that support transactions between banks. An acquirer maps the iKP protocol conducted on the Internet to the protocols utilized on the financial networks. iKP requires no changes in the communication between the issuer and acquirer banks.

Communications between buyer and seller are assumed to occur over a public network such as the Internet. iKP is specifically designed to address security issues that arise in this environment.

Communications between seller and acquirer may be via the Internet or over private channels. iKP may be used in either case, but it specifically provides for security concerns that arise when the Internet is also utilized for this purpose.

Secure financial networks already exist to connect acquirers to issuers. Consequently, iKP assumes that adequate security is already in place between these parties.

5 High-Level Protocol Description

ZIP includes three protocol scenarios:

1. Payment Authorization (with cancellation option)

2. Payment Clearance (Capture)[1]
3. Inquiry

This protocols are summarized below. Message fields and detailed descriptions thereof are discussed in later sections.

5.1 Payment Authorization

It is assumed that, prior to invoking iKP, the buyer and seller have agreed on the Purchase Order details, price/currency and the payment method.

```
+--------+                    +--------+                   +--------+
| Buyer  |                    | Seller |                   |Acquirer|
+--------+                    +--------+                   +--------+

             INITIATE
-------------------------------->
             INVOICE
<--------------------------------
             PAYMENT
-------------------------------->
             CANCEL (optional)
<- - - - - - - - - - - - - - - -
                                          AUTH-REQUEST
                                     -------------------------->
                                          AUTH-RESPONSE
                                     <--------------------------
             CONFIRM
<--------------------------------
```

This is the basic payment protocol. The seller may choose to combine payment authorization with payment clearing in which case the present protocol suffices. Alternatively, the seller may decide to only authorize payment and perform the actual clearance/capture function at some later time.

Regardless of the acquirer's decision in AUTH-RESPONSE, the seller sends a CONFIRM (even if the response is negative) to the buyer.

If the seller chooses to (or is forced to) delay contacting the acquirer, he can send a STATUS flow (see below) to the buyer after receiving PAYMENT. This is to keep the buyer abreast of the transaction status. Alternatively, the seller can elect to take the risk and send a CONFIRM to the buyer without having any real contact with the acquirer.

In the event that the seller is unable or unwilling, for some reason, to process the buyer's payment, the payment authorization protocol may be truncated (terminated) with a CANCEL flow even before trying to contact the acquirer.

5.2 Separate Payment Clearance/Capture

At the discretion of the seller, payment clearance may be performed either as part of authorization, or postponed until later. This protocol supports delayed/separate clearance. (Of course, an acquirer may dictate its policy on this subject to all constituent sellers.)

Multiple clearance flows against the payment authorization are supported.

[1] The terms "clearance" and "capture" are used interchangeably throughout this document.

```
+--------+                    +--------+                +--------+
| Buyer  |                    | Seller |                |Acquirer|
+--------+                    +--------+                +--------+

   (after previous authorization)

                                           CLRN-REQUEST
                                    -------------------------->
                                           CLRN-RESPONSE
                                    <--------------------------
```

5.3 Refunds

Sellers may issue refunds for previously cleared payments. Although it is understood that refunds are typically triggered by consumers/buyers, the interaction between buyer and seller that leads to an eventual refund is assumed to take place off-line (i.e., outside iKP/ZIP.)

Within iKP/ZIP, a refund transaction – for all practical purposes – is equivalent to (and treated as) a clearance/capture transaction. This is mainly because a refund is, essentially, a clearance with the negative amount. The difference between a refund and a clearance manifests itself only within the domain of the financial clearing network.

5.4 Inquiry

```
+--------+                    +--------+                +--------+
| Buyer  |                    | Seller |                |Acquirer|
+--------+                    +--------+                +--------+

             INQUIRY
--------------------------------->
     STATUS, CONFIRM or CANCEL
<---------------------------------
```

The buyer can ask the seller about the status of a specific payment. The buyer may transmit INQUIRY at any time after submitting a PAYMENT flow. The seller must be able to respond for some time after the payment transaction is completed; the exact time period is the choice of the seller or may be specified by financial institutions.

6 Detailed Protocol Description

6.1 Atomic Fields/Symbols

```
 A, S, B     |    Protocol Participants: Acquirer, Seller, Buyer
 [...]       |    Optional fields
 PFLAGS      |    Protocol option flags (see below.)
 STATE       |    Transaction state (see INQUIRY flow.)
 PKE_Z/SKE_Z |    Z's public/private key-pair used for encryption only;
             |    Z in {A,S,B}
 PKS_Z/SKS_Z |    Z's public/private key-pair used for signatures only;
             |    Z in {A,S,B}
 E_A(TXT)    |    Encryption (with integrity) under acquirer's public
```

```
             |    key PKE_A. See section 10.5 for details.
S_Z(TXT)     |    Signature under Z's private key SKS_Z; Z in {A,S,B}
             |    NOTE: S_Z(TXT) does NOT include TXT in the clear
SALT_B       |    Random number generated by B; used to ensure privacy
             |    of order information on the S<->A link; also used
             |    to provide freshness of SIG_S and SIG_A.
V            |    Random number generated by the seller; used
             |    to bind CONFIRM and INVOICE messages.
VC           |    Random number generated by the seller; used
             |    to bind CONFIRM/CANCEL and INVOICE messages.
AUTH_PRICE   |    Amount and currency code authorized.
CLRN_PRICE   |    Amount and currency code cleared; may differ from AUTH_PRICE.
AUTH_TIME    |    Timestamp of payment authorization; set by acquirer
CLRN_TIME    |    Timestamp of payment clearance; set by acquirer
CLRN_SEQ     |    Clearance sequence number set by seller. Initially set to 0
             |    and incremented for each successive clearance/refund.
DATE         |    Seller's current date/time;
NONCE_S      |    Seller's nonce used as a one-time challenge.
             |    NONCE_S should be unique and random.
ID_S         |    Seller id. This identifies seller to acquirer.
H(text)      |    Strong one-way hash function applied to "text"; e.g., MD5.
DESC         |    Purchase details as an opaque string.  This defines
             |    the agreement between the buyer and seller as to
             |    what is being paid for in this payment transaction.
             |    Not explicitly carried in iKP flows.
BAN          |    Buyer's Account Number (Eg. credit card No.)
PIN          |    Optional buyer's PIN
R_B          |    Random number chosen by B to form ID_B. It must be
             |    random (not just unique) in order to serve as proof
             |    by the buyer that the seller agreed to the payment.
             |    See section on disputes below.
SALT_C       |    Random number used to salt the account number in the
             |    buyer's certificate.
EXPIRATION   |    Expiration date of the buyer's account.
INVOICE_EXP  |    Invoice (offer) expiration specified by the seller.
ID_B         |    One-time buyer ID which uniquely identifies B;
             |    computed as ID_B=H(R_B,BAN).
RESP_CODE    |    Authorization or capture response code from the acquirer.
             |    Can also be set by the seller in case of a cancellation.
OPT_SIG_Z    |    Optional text for inclusion in signatures; Z in {A,S,B}
             |    Not explicitly carried in iKP flows.
TID_Z        |    Transaction id assigned be each party to an iKP transaction
             |    Generated at a layer above iKP and not explicitly carried
             |    in iKP flows (higher layers transport TIDs.)
```

6.2 Composite Fields/Symbols

```
COMMON       |    "information held in common by all parties:"
             |    TID_S, TID_B, PFLAGS, AUTH\_PRICE, ID_S, DATE,
             |    INVOICE_EXP, NONCE_S, ID_B, H(DESC,SALT_B), H(V), H(VC)
CLEAR        |    "information transmitted in the clear"
             |    PFLAGS, ID_S, DATE, NONCE_S, H(COMMON),
             |    INVOICE_EXP, H(DESC,SALT_B), H(V), H(VC)
```

```
SLIP         |    "payment instructions/slip"
             |    AUTH_PRICE, H(COMMON), BAN, R_B, EXPIRATION, [SALT_C or PIN]
SIG_S        |    "seller's signature"
             |    S_S( H(COMMON), OPT_SIG_S )
SIG_B        |    "buyer's signature"
             |    S_B( E_A(SLIP), H(COMMON), OPT_SIG_B )
SIG_A        |    "acquirer's signature"
             |    S_A( RESP_CODE, AUTH_TIME, H(COMMON), OPT_SIG_A )
SIG_S_CLRN   |    "seller's clearance signature"
             |    S_S( H(COMMON), CLRN_PRICE, OPT_SIG_S_CLRN, CLRN_SEQ )
SIG_A_CLRN   |    "acquirer's clearance signature"
             |    S_A( RESP_CODE, CLRN_TIME, CLRN_PRICE, CLRN_SEQ,
             |    H(COMMON), OPT_SIG_A_CLRN )
```

6.3 Typing of Message Fields

Every signature type generated in iKP is assigned a unique signature identifier. Every signature operation (generation/verification) automatically includes a signature identifier for a specific signature type. The same holds for all hash function computations.

The following distinct signature types are identified: SIG_B, SIG_S, SIG_A, SIG_S_CLRN, SIG_A_CLRN.[2]

The following hash function computations are uniquely identified: H(COMMON), H(V), H(VC), H(R_B,BAN) and H(DESC,SALT_B).

6.4 Opaque Fields

Some of the aforementioned protocol fields are treated opaquely by iKP. "Opaque" in this context means that these flows are not carried within iKP protocol messages. At the same time, these fields are authenticated and integrity-protected by iKP. Some of these fields may (and sometimes have to) be tacked on by the higher-layer software. These fields are:

- DESC
  Purchase details (e.g., merchandise description) may have to be explicitly transmitted between buyers and sellers. However, it is not recommended for transmission to the acquirer.
- TID_S and TID_B
  Seller's and buyer's transaction identifiers are generated outside of iKP (by the higher-layer software). By virtue of being part of COMMON they are integrity-protected by iKP. Details of their transport is left unspecified.
- All fields of the form OPT_SIG_Z are (as the name suggests) optional. For example, they can be used to carry credentials/certificates of various entities or periodic account statements.

6.5 Flow Definitions

6.5.1 Payment with In-line Authorization Flow

This is the basic payment authorization protocol. One of the protocol option flags (in CLEAR) indicates whether the seller wishes the acquirer to perform payment clearance at the same time. The response code from the acquirer indicates whether authorization is given, and (if clearance was requested) whether the payment was cleared.

[2] Since multiple clearance transactions against the same payment authorization are allowed, the clearance signatures (SIG_S_CLRN, SIG_A_CLRN) are further distinguished by *CLRN_SEQ*.

```
+--------+                      +--------+                 +--------+
| Buyer  |                      | Seller |                 |Acquirer|
+--------+                      +--------+                 +--------+

           PFLAGS, SALT_B, ID_B
--INITIATE------------------------->

              CLEAR, [SIG_S]
<-INVOICE---------------------------

           E_A(SLIP), [SIG_B]
--PAYMENT-------------------------->

       RESP_CODE, VC, [SIG_S]
<-CANCEL- - - - - - - - - - - - - -
                                        CLEAR, E_A(SLIP), SIG_S, [SIG_B]
                                   -AUTH-REQUEST---------------------->

                                        RESP_CODE, AUTH_TIME, SIG_A
                                   <-AUTH-RESPONSE---------------------
 RESP_CODE (=positive), AUTH_TIME,
        V, [SIG_S], [SIG_A]
<-CONFIRM---------------------------


 RESP_CODE (=negative), AUTH_TIME,
        VC, [SIG_S], [SIG_A]
<-CONFIRM---------------------------
```

Note that the CANCEL flow is only used if the seller decides – for whatever reason – not to go ahead payment authorization. CANCEL may not be used if the seller has received an AUTH-RESPONSE.

If the payment clearance is not done (either because it was not requested by the seller, or because the acquirer was unable to perform it), the seller must subsequently either transmit the CLRN_REQUEST message or take further processing of this payment off-line.

Upon receipt of a CONFIRM flow, The combination of SIG_A, SIG_S and V affords the buyer with a proof that the payment transaction has completed and the seller has committed to the transaction.

6.5.2 Clearance Flow

```
+--------+                      +--------+                 +--------+
| Buyer  |                      | Seller |                 |Acquirer|
+--------+                      +--------+                 +--------+
                                         CLRN_PRICE, CLRN_SEQ,
                                                SIG_S_CLRN
                                  --CLRN-REQUEST------------------->

                                             RESP_CODE, CLRN_TIME,
                                             CLRN_SEQ, SIG_A_CLRN
                                  <-CLRN-RESPONSE------------------
```

The seller uses the clearance flow to request the actual transfer of funds from the buyer to the merchant. This flow may only occur after the corresponding authorization flow if clerance was not performed as part of payment authorization.

The seller must send the CLRN-REQUEST message to the same acquirer used for payment authorization.

6.5.3 Inquiry Flow

```
+--------+                     +--------+                  +--------+
| Buyer  |                     | Seller |                  |Acquirer|
+--------+                     +--------+                  +--------+

             H(COMMON)
--INQUIRY-------------------------->

           (same as above)
<-CONFIRM---------------------------

               - OR -

           (same as above)
<-CANCEL----------------------------

               - OR -

               STATE
<-STATUS----------------------------
```

The buyer may transmit INQUIRY at any time after sending the PAYMENT flow. The response from the seller can be either CONFIRM (if the seller has received AUTH-RESPONSE) or STATUS (if not). In the latter case, STATE represents the current transaction state from the seller's perspective.

6.6 Protocol Options and Flags

The payment authorization protocol may be suspended after the second (INVOICE) flow. This abbreviated version can be used for gathering SIG_S-s (i.e., signed invoices) from multiple sellers for the purpose of browsing and comparative shopping. The abbreviated protocol run can be resumed at a later time provided that SIG_S is still timely/valid.

In addition, ZIP supports the following transaction options:

SIG_B Buyer's signature SIG_B in PAYMENT and AUTH-REQUEST. While this option is at the discretion of the buyer, a seller can refuse to issue an INVOICE if it is the seller's policy to alway require SIG_B and the buyer is not able to provide it.

IMPORTANT: SIG_B must be fixed for a given buyer-account combination. In other word, a buyer who has an ability to generate signatures must always do so. However, it is ultimately the acquirer's responsibility to make sure that a buyer with a signature capability always uses the SIG_B option.

SIG_S Sellers's signature SIG_S in INVOICE. This option is set by the buyer but, as before, a given seller can refuse to comply because, for example, it is not interested in giving out signed "offers" for buyers that aren't ready to pay.

CONFIRM Indicates that CONFIRM is requested. It is set by the BUYER; it is envisaged that every seller should support this option.

SIG_A SIG_A is requested in CONFIRM (NOTE: this option can only be used in conjunction with the CONFIRM option.) Set by the buyer.

CLRN Authorization and clearance (capture) are performed together. This one is set by the seller. While a buyer may, in principle, refuse it, it is not likely...

no_Enc Buyer does NOT use encryption; SLIP is sent in the clear. This flag is set by the buyer. Sellers have no say over this option. (IMPORTANT: this option can only be used in conjunction with the SIG_B option.) The purpose of this option is to avoid the expense of encryption and to satisfy certain export regulations in cases when BANs are not treated as secret or sensitive information.

7 Protocol Flow Processing

This section describes normal protocol operation by all parties. Handling of errors and other exceptions is discussed in the next section.

All parties involved are assumed to have access to stable, non-volatile storage. The term "recording" means commitment to stable storage. It is further assumed that the buyer and seller commit all local variables – transaction ids, nonces, signatures, etc. – to stable storage BEFORE sending out a message flow.

Moreover, it is assumed below that incoming flows are associated (matched) with outstanding, currently-active transaction by the transaction layer software, i.e., software residing above iKP message processing code.

7.1 Payment Authorization Protocol

This section discusses how basic payment authorization is handled by all participants.

7.1.1 INITATE Composition

The buyer transmits the INITIATE message at the end of the negotiation phase and the start of the payment phase; this message delimits the boundary between the two phases.

Note that DESC (purchase details) and AUTH_PRICE (purchase amount) are either agreed upon a-priori or communicated in-band alongside INITIATE or INVOICE flow. In any case, DESC is not carried within any iKP flow.

The buyer forms INITIATE as follows:

1. produces ID_B by generating random number R_B and computing ID_B = H(R_B,BAN).
2. generates another random number SALT_B to be used for "salting" the hash of the purchase description (DESC) in subsequent messages; also used as a challenge to the seller.
3. sets PFLAGS to reflect desired protocol options.
4. sends INITIATE.

7.1.2 INITIATE Processing and INVOICE Composition

Upon receipt of the INITIATE flow, the seller performs the following steps:

1. checks buyer options set in PFLAGS and returns an error if it finds it incompatible with its own policy. Otherwise, the seller sets the CLRN option flag in PFLAGS to reflect its payment processing policy.

2. gets a clock reading and sets DATE.
3. generates nonce NONCE_S to be used later for freshness/uniqueness checks.
4. computes H(DESC,SALT_B)
5. generates random [V,VC] pair and computes the corresponding [H(V),H(VC)] vector.
6. encodes/linearizes COMMON as defined above and computes H(COMMON). NOTE: the seller does not need to additionally "salt" H(COMMON) because it contains the already-salted H(DESC,SALT_B) as well as NONCE_S..
7. computes SIG_S if SIG_S flag is set.
8. sends INVOICE.

7.1.3 INVOICE Processing

Upon receiving the INVOICE the buyer takes the following steps:

1. retrieves the fields in CLEAR.
2. validates DATE (within an implementation-defined skew).
3. computes H(DESC, SALT_B) and compares it with H(DESC, SALT_B) from CLEAR. This confirms that the buyer and seller agree on the purchase details.
4. computes H(COMMON) and compares it with H(COMMON) from CLEAR. This confirms that the buyer and seller agree on information contained in CLEAR, in particular AUTH_PRICE.
5. verifies SIG_S if SIG_S option is set.

The protocol may terminate at this point since, sometimes, the buyer may choose not to proceed with the actual payment. Seller keeps transaction state for some implementation-specific period of time (perhaps set to the maximum time skew for a given acquirer.) Thereafter, the transaction is deleted.

7.1.4 PAYMENT Composition

The buyer performs the following steps when it is ready to proceed with the payment:

1. if no_Enc flag is not set, forms SLIP and encrypts it under PKE_A as described in "Use of ENCRYPTION in iKP/ZIP" below.
2. otherwise, SLIP is simply linearized (encoded) in the clear.
3. if SIG_B is set, computes SIG_B.
4. sends PAYMENT.

7.1.5 PAYMENT Processing

Upon receiving PAYMENT the seller first does the following:

1. verifies SIG_B if SIG_B is set and buyer's credentials are available. (NOTE: if buyer's credentials are not available, the seller may decide to either cancel the transaction or go ahead with it. In case of the former, a CANCEL message is sent to the buyer.)

2. chooses whether to perform immediate or delayed authorization. If the former, proceeds to AUTH-REQUEST message processing. Otherwise, seller returns STATUS to the buyer.
3. if, for any reason, the seller decides not to process the payment further, it can generate and return a CANCEL message to the buyer.

7.1.6 AUTH-REQUEST Processing

The seller performs the following steps when it is ready to obtain payment authorization:

1. if SIG_S was computed for INVOICE, copies the same SIG_S into AUTH-REQUEST.
2. if SIG_S is not already computed, computes SIG_S and places it in AUTH-REQUEST
3. sends AUTH-REQUEST.

When the acquirer gateway receives AUTH-REQUEST, it:

1. checks that the DATE is valid (within an implementation-defined skew).
2. verifies SIG_S
3. decrypts E_A(SLIP) and obtains:
   AUTH_PRICE, BAN, R_B, EXPIRATION, (optional) PIN and H(COMMON).
4. verifies SIG_B if present. (Note that SIG_B must always be present if SIG_B option flag is set and if the buyer has SIG_B capability.)
5. encodes/linearizes COMMON; then, computes H(COMMON) and cross-checks it with both H(COMMON) in CLEAR and H(COMMON) in SLIP.
6. validates that R_B and BAN match ID_B found in CLEAR.
7. composes and sends authorization request on the financial network. If capture is requested in PFLAGS, includes a capture request with the authorization request.
8. proceeds to AUTH-RESPONSE once the response is received from the the financial network.

Note that the gateway and/or financial network MUST perform replay detection only if the seller requests payment capture processing. This is to ensure that payments are not charged to buyers multiple times. Replay detection for authorization-only requests is a policy matter determined by the individual payment system providers.

7.1.7 AUTH-RESPONSE Processing

When the acquirer gateway receives an authorization (and possibly capture) response from the financial network, it:

1. sets RESP_CODE to indicate any of:
   - authorization denied.
   - authorization approved but payment not captured.
   - authorization approved and payment captured (only if capture requested in PFLAGS).
2. sets OPT_SIG_A to the authorization code and other optional data provided by the financial network Data in OPT_SIG_A, while included in the computation of SIG_A, is treated opaquely by iKP and is not carried in the iKP messages. (Its transport is left up to the upper layer.)

3. computes SIG_A.

4. sends AUTH-RESPONSE to the seller.

When the seller receives AUTH-RESPONSE, it:

1. verifies SIG_A

2. records RESP_CODE, AUTH_TIME and OPT_SIG_A

3. if RESP_CODE is positive, generates positive CONFIRM by including V in the message.

4. if RESP_CODE is negative, generates negative CONFIRM by including VC in the message.

5. CONFIRM is sent only if CONFIRM flag is set

7.1.8 CONFIRM Processing

The seller generates CONFIRM as follows:

1. copies RESP_CODE and AUTH_TIME from AUTH-RESPONSE

2. if SIG_A flag is set, copies SIG_A (note that OPT_SIG_A has to be tacked on above iKP)

3. copies SIG_S from AUTH_REQUEST (if not provided in INVOICE)

4. includes V (or VC) as evidence to the buyer that the payment transaction has completed.

5. sends CONFIRM.

This completes the payment transaction from the seller's perspective. The only way the seller communicates with the buyer again about the same transaction is if the buyer sends INQUIRY.

When the buyer receives CONFIRM, it:

1. verifies SIG_A

2. if SIG_S flag is not set, verifies SIG_S

3. verifies that V hashes to H(V)

4. records RESP_CODE, V, SIG_A, and SIG_S as evidence of the transaction's completion

7.1.9 CANCEL Processing

The seller generates CANCEL as follows:

1. sets RESP_CODE to a specific error code (e.g., merchandise out of stock.) Note that the meaning of RESP_CODE in CANCEL is different from that in AUTH-RESPONSE.

2. if SIG_S is not previously computed, computes SIG_S afresh

3. includes VC as evidence to the buyer that the payment transaction has been aborted, i.e., the seller promises not to pursue buyer's payment.

4. sends CANCEL

When the buyer receives CANCEL, it:

1. if SIG_S flag is not set, verifies SIG_S
2. verifies that VC hashes to H(VC)
3. stores RESP_CODE, VC, and SIG_S as evidence of the transaction's dismissal.

Note that any CANCEL received after a valid CONFIRM should be ignored.

7.1.10 STATUS Processing

When the buyer receives STATUS, it:

1. stores the current payment status (from the seller's perspective) reflected in STATE.

7.2 Clearance (Capture) Protocol

The seller utilizes the clearance protocol when doing on-line (rather than batch) payment capture processing, and the RESP_CODE received in AUTH-RESPONSE indicates that payment is authorized but not captured. Note that clearance must be performed with the same acquirer that handled previous authorization. Furthermore, clearance cannot be initiated before AUTH-RESPONSE is received.

Moreover, as mentioned earlier, the clearance protocol may be used for implementing seller-initiated refunds.

7.2.1 CLRN-REQUEST Processing

The seller performs the following steps:

1. obtains CLRN_PRICE
2. computes SIG_S_CLRN
3. increments CLRN_SEQ
4. sends CLRN-REQUEST.

When the acquirer gateway receives CLRN-REQUEST, it:

1. checks that the transaction is in the appropriate state, i.e., that either a) payment authorization or b) payment clerance, has been performed.
2. examines previous clearance transactions against the same payment authorization and checks that CLRN_SEQ is new (i.e., greater than that in the last CLRN-REQUEST processed.)
3. verifies SIG_S_CLRN

Note that the acquirer gateway and/or financial network MUST perform replay detection of capture requests. This is to ensure that payments are not charged to buyers multiple times.

The relationship between AUTH_PRICE and CLRN_PRICE (as well as that between CLRN_PRICE and earlier cleared amounts) is not checked within iKP. The difference between the two values is subject to locally-defined constraints. (Checking is assumed to be done at the transaction layer or higher.)

7.2.2 CLRN-RESPONSE Processing

When the acquirer gateway receives the appropriate response from the financial clearing network, it:

1. sets RESP_CODE to indicate any of:
   - capture denied
   - capture completed
2. sets time-of-clearance, CLRN_TIME
3. computes SIG_A_CLRN
4. sends CLRN-RESPONSE to the seller

When the seller receives CLRN-RESPONSE, it:

1. verifies SIG_A_CLRN
2. archives RESP_CODE and other fields for use in any future STATUS or CONFIRM message.

7.3 INQUIRY Processing

The buyer can issue INQUIRY at any time after sending the PAYMENT flow. Composing INQUIRY does not require any special actions since its contents are limited to H(COMMON).

When the seller receives an INQUIRY flow he – depending on the current state of the transaction – transmits CONFIRM, CANCEL or STATUS to the buyer.

8 Fault Tolerance and Exception Handling

As can be expected in any communication environment – especially, in the Internet – absolute reliability is next to impossible. Therefore, in order to design, not only secure, but also robust, payment protocols, we need to consider all possible anomalous scenarios.

ZIP makes no assumptions about the robustness of the underlying network infrastructure since it is envisaged that iKP will operate in environments with widely varying degrees of reliability.

It is assumed that the communications software (at the transaction layer) correlates successive protocol messages of one payment transaction to each other and to the internal state of each participant.

It is assumed that all parties in iKP (except acquirer) implement timeouts and retransmissions whenever a message elicits no reply. This is one of the functions of the transaction layer.

All unexpected messages, i.e., those not corresponding to an outstanding or recorded transaction, must be ignored.

All otherwise invalid messages (e.g., acquirer receiving INITIATE) are similarly ignored.

Duplicate messages must be detected by the transaction layer. Transaction ids, TID_B and TID_S as well as message type identifiers should be used for that purpose. In the event that a duplicate message is, nonetheless, received by the ZIP iKP API, an error indicating "invalid state" is returned.

All protocol parties are assumed to have access to stable, non-volatile storage. The term "recording" is used to mean commitment to stable storage.

9 Rationales and Explanations

This section explains some of the features of the protocol design.

9.1 Performance

Cryptographic operations such as computation/verification of public key signatures and en/de-cryption are computationally expensive. iKP and ZIP are designed to improve performance by minimizing the number of cryptographic operations.

9.2 Verification of CONFIRM and CANCEL

The purpose of H(V),H(VC) in the CLEAR part of the INVOICE message and of V or VC in CONFIRM, or VC in CANCEL is to give the buyer proof of the seller's commitment as to the fate of the transaction. This is done without requiring any additional signatures from the seller. Possession of V (or VC) by the buyer shows that the seller must have sent CONFIRM (or CANCEL) to the buyer.

9.3 Adherence to Export Rules

The protocol is designed to minimize the amount of data encrypted, in order to satisfy the export control rules of the U.S. government. As described in detail in the next section, only SLIP is encrypted, and the data therein is limited to financial information.

9.4 Use of ENCRYPTION in iKP

iKP requires the buyer to encrypt the SLIP as part of the PAYMENT message, and optionally in the AUTH-RESPONSE message. (The sole exception to this is when both the "no_Enc" and the "SIG_B" option flags are set.)

RSA encryption is used in conjunction with integrity techniques described in [6] and PKCS encoding.

Encryption is ALWAYS performed using the encryption public key of the acquirer – PKE_A – for the PAYMENT message. It is assumed that PKE_A has a modulus size of at least 1024 bits. The format of the linearized (encoded) SLIP to be encrypted is as follows:

```
ENC_SLIP = [ AUTH_PRICE, H(COMMON), BAN, R_B, PIN, EXPIRATION, PADDING]
```

Fields within ENC_SLIP are described under the heading "Atomic Fields/Symbols", above. The sizes of these fields are as follows:

AUTH_PRICE —- 64 bits

BAN —- 0-128 bits; (128 bits can be used to encode up to 38 decimal digits. (For example, current credit cards are only 12 digits.)

EXPIRATION —- 32 bits

PIN —- 0-64 bits (up to 19 decimal digits)

H(COMMON) —- 128 bits

R_B —- At least 128 bits

PADDING —- Padding, at least 64 bits long; maximum length is the difference between 896 bits and the sum (in bits) of all previous fields.

All of the above components are encoded into a 896-bit long string ENC_SLIP. The actual encryption process is adapted from [6] and proceeds as follows:

1. Prepend 64 bits of zeros to ENC_SLIP to form x=[0^{64},ENC_SLIP] (x is 960 bits long.)
2. Generate random 64-bit string E_SALT.
3. Compute a = x XOR H_1(E_SALT) and then let b = E_SALT XOR H_2(a) H_1() is a hash function outputting 960 bits and H_2() is a hash function outputting 64 bits. Example ways of constructing H_1 and H_2 from H are given below.
4. Compute E_a(a,b) (combined length of a,b is 1024 bits).

Decryption proceeds as follows:

1. Compute D_a(Ea(a,b))=a,b.
2. Compute E_SALT = b XOR H_2(a).
3. Compute x = a XOR H_1(E_SALT).
4. Check for existence of 64 leading 0-s in x and obtain E_SLIP.

The hash functions H_1 and H_2 are computed as:

- H_2(DATA) = First 64 bits of output of H(DATA).
- H_1(DATA) = First 960 bits of the sequence H(START,0,DATA), H(START,1,DATA),H(START,2,DATA),... where START is a constant (set to zero in ZIP.)

References

[1] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik and Michael Waidner, *"iKP - A Family of Secure Electronic Payment Protocols"*, Usenix Electronic Commerce Workshop, July 1995.

[2] M. Linehan and G. Tsudik, *"Internet Keyed Payments Protocol (iKP)"*, Internet Draft, "draft-tsudik-ikp-00.txt", June 1995.

[3] S. Larsen, *"iKP Transaction Layer Functional Specification"*, Working document, available from lst@zurich.ibm.com or gts@zurich.ibm.com.

[4] K. Hickman, *"The SSL Protocol"*, Internet Draft, "draft-hickman-netscape-ssl-00.txt", April 1995.

[5] E. Rescorla and A. Schiffman, *"The Secure HyperText Transfer Protocol"*, Internet Draft "draft-rescorla-shttp-0.txt", December 1994.

[6] M. Bellare and P. Rogaway, *"Optimal Asymmetric Encryption"*, Eurocrypt 94.

[7] R. Atkinson, *"Security Architecture for the Internet Protocol"*, Internet Draft "draft-ietf-ipsec-arch-02.txt", work in progress, May 11, 1995.

[8] R. Rivest, *"The MD5 Message-Digest Algorithm"*, Internet RFC 1321, April, 1992.

10 Editor's Address


```
Gene Tsudik
IBM Zurich Research Laboratory
Saeumerstrasse 4
CH-8803, Rueschlikon, Switzerland
gts@zurich.ibm.com
```

A iKP Core API

This appendix contains some notes pertaining to iKP Core API – a set of functions providing a low-level interface for constructing and processing iKP protocol flows.

In general, this API is not intended for use by application programs. Its primitives are callable by the iKP transaction layer (described in [3].)

A.1 iKP Core API Return/Error Codes

All iKP API functions return a "major" error code. An error code of zero indicates (as is common in C) successful completion. A complete listing of all major return codes follows:

1. *ikp_failed* Miscellaneous failure...
2. *noMemory* Memory allocation problem; most probably *malloc()* failed.
3. *cryptoError* Error in CRYPTO API call, *minor_rc* returns the actual CRYPTO error code.
4. *programmingError*
5. *outdated* Expired date/timestamp in invoice or slip.
6. *content_mismatch*
7. *encError* Error in one the encoding routines.
8. *decErrorMalformed* Decoding error; malformed data stream.
9. *decErrorBadMsgtype* Unexpected message type found upon decoding.
10. *ReadError* One or more of the input parameters passed to an API function is invalid (NULL).
11. *WriteError* One or more of the output parameters (e.g., minor_rc) passed to an API function is invalid (NULL).
12. *ReadError1* One or more of the input parameters (which is itself a pointer) is invalid (NULL).
13. *WriteError1* One or more of the output parameters (which is itself a pointer) is invalid (NULL).
14. *optionMismatch* Protocol options specified by the buyer conflict with seller's policy or vice-a-versa.
15. *sigMissing* A signature is missing from an incoming flow, e.g., SIG_S is missing in AUTH_REQUEST.
16. *WrongState* An API function called while a transaction context is in the wrong state; e.g., a function to process the seller's invoice is called after the payment has been already made.
17. *sellerMismatch* Seller id in an incoming flow does not match its previously stored counterpart.
18. *fileError* File I/O error; never occurs within iKP API.
19. *decErrorBadVersionNo* Wrong iKP version number upon decoding.
20. *encErrorBadMsgType* Invalid iKP message type identifier encountered upon encoding.
21. *encErrorBadVersionNo* Wrong iKP version number upon encoding.
22. *hashVmismatch* H(V) or H(VC) – depending on the context – does not match the previously stored H(V) or H(VC). Currently, this error can only occur at the buyer.
23. *WrongRole* A function is called by a principal of the worng type; e.g., buyer tries to call an API function that composes an INVOICE flow.

24. *ErrorPinTooLong* A PIN longer than the current maiximum (64 bits at the moment) has been specified.

25. *ErrorBanTooLong* A PIN longer than the current maiximum (128 bits at the moment) has been specified.

In addition to the above, all iKP API functions require an output parameter "minor_rc". It is used to return specific return/error codes that stem from within different code components. In particular, this parameter reflects the CRYPTO API error codes in the event of an error in one the CRYPTO API functions, e.g., invalid key passed to the signature verification function.

A.2 Input and Output Tokens

iKP protocol flows are linearized (encoded) by appropriate iKP functions. Each function that processes a particular iKP flow requires *input_token* as one of the parameters.

input_token is a an opaque string – a pointer to a structure containing the length of the data and a pointer to the stream of data bytes. Both *input_token* itself and the data pointer within it must always be defined (i.e., non-NULL) in all API functions. Otherwise, a *ReadError* or *ReadError1* is returned.

Similarly, all iKP functions that result in a construction of an iKP protocol flow require *output_token* as one of the parameters.

output_token is also an opaque string – a pointer to a structure containing the length of the data and a pointer to the stream of data bytes. The *output_token* itself must always be defined (i.e., non-NULL) in all API functions. Otherwise, a *WriteError* is returned.

A.3 Protocol Option Flags

Current implementation of iKP (ZIP) supports the following protocol options:

SIG_B - Buyer signature offered in Payment flow

SIG_S - Seller signature requested in Invoice flow

CONFIRM - Confirm flow requested

NO_ENC - Encryption not used in PAYMENT flow, i.e., SLIP is not encrypted. **NOTE:** this flag is mutually exclusive with SIG_B)

SIG_A - Include Acquirer's signature in CONFIRM flow (this flag would indicate that SLIP is not encrypted; NOTE: this flag is mutually exclusive with SIG_B)

CLRN - Clearing/capture is performed together with authorization

CLRN is the only option chosen/set by the seller. All other flags are under control of the buyer, however, a seller can refuse to honor them.

A.4 Buyer API Functions

This section lists API functions callable on the buyer side. Invocation of these functions by acquirers and sellers causes an error – *WrongRole*; see above.

```
ikp_error_code_t ikp_Initiate_payment  (
ikp_flags_t                      pflags,         /* protocol flags */
ikp_data_t                       *TID_B,         /* buyer's TID  (input) */
ikp_price_t                      *amount,        /* currency and amount */
ikp_data_t                       *desc,          /* what is being purchased */
ikp_seller_name_t                *seller_name,   /* name of the seller */
ikp_acct_priv_creds_t            *my_acct_creds, /* account info */
ikp_buyer_ctx_t                  **trans_ctx_p,  /* new context pointer */
ikp_data_t                       *output_token,  /* INITIATE flow */
ikp_error_code_t                 *minor_rc       /* minor error code */
      );

ikp_error_code_t ikp_Process_invoice            (
ikp_data_t                       *TID_S,         /* seller's TID  */
ikp_buyer_ctx_t                  *trans_ctx,     /* seller's context */
ikp_data_t                       *input_token,   /* INVOICE or ERROR flow */
ikp_creds_t                      *s_creds,       /* seller's credentials */
ikp_data_t                       *opt_sig_S,     /* optional data for SIG_S */
ikp_data_t                       *output_token,
ikp_error_code_t                 *minor_rc
                                      );

ikp_error_code_t ikp_Submit_payment (
ikp_buyer_ctx_t                  *ctx,
ikp_acct_priv_creds_t            *my_acct_creds, /* account info for ctx */
ikp_time_t                       payment_exp,    /* payment valid 'till? */
ikp_creds_t                      *a_creds,       /* acquirer's credentials */
ikp_data_t                       *opt_sig_B,     /* opt. for SIG_B */
ikp_data_t                       *output_token,
ikp_error_code_t                 *minor_rc
);

ikp_error_code_t ikp_Process_confirm (
ikp_buyer_ctx_t                    *trans_ctx,
ikp_data_t                         *input_token,   /* CONFIRM flow */
ikp_creds_t                        *a_creds,       /* acquirer's creds */
ikp_creds_t                        *s_creds,       /* seller's creds */
ikp_data_t                         *opt_sig_S,     /* opt. data for SIG_A */
ikp_data_t                         *opt_sig_A,     /* opt. data for SIG_S */
ikp_error_code_t                   *minor_rc
                                        );

ikp_error_code_t ikp_Process_cancel       (
ikp_buyer_ctx_t                    *ctx,
ikp_creds_t                        *s_creds,      /* seller's credentials */
ikp_data_t                         *opt_sig_S,    /* optional data for SIG_S*/
ikp_data_t                         *input_token,  /* CANCEL flow         */
ikp_error_code_t                   *minor_rc
                                     );
```

```
ikp_error_code_t ikp_inquiry           (
ikp_buyer_ctx_t                  *ctx,
ikp_data_t                       *output_token,
ikp_error_code_t                 *minor_rc
                                       );

ikp_error_code_t ikp_Process_status    (
ikp_buyer_ctx_t                  *ctx,
ikp_data_t                       *input_token,   /* STATUS flow */
ikp_protocol_state_t             *state,
ikp_error_code_t                 *minor_rc
                                       );
```

A.5 Seller Functions

This section lists API functions callable on the seller side. Invocation of these functions by buyers and acquirers causes an error – *WrongRole*; see above.

```
ikp_error_code_t  ikp_Process_initiate  (
ikp_flags_t                      pflags,         /* auth/clear flags ???? */
ikp_data_t                       *TID_B,         /* buyer's TID  */
ikp_data_t                       *TID_S,         /* seller's TID  */
ikp_seller_ctx_t                 **trans_ctx_p,
ikp_acct_priv_creds_t            *my_acct_creds, /* account info for ctx */
ikp_data_t                       *input_token,   /* INITIATE flow */
ikp_price_t                      *amount,        /* expected price */
ikp_data_t                       *desc,          /* what is being sold */
ikp_time_t                       invoice_exp,    /* invoice valid until? */
ikp_data_t                       *opt_sig_S,     /* optional data for SIG_S */
ikp_data_t                       *output_token,  /* INVOICE or ERROR flow */
ikp_error_code_t                 *minor_rc
                                       );

ikp_error_code_t  ikp_Process_payment  (
ikp_seller_ctx_t                 *ctx,           /* seller's context */
ikp_data_t                       *input_token,   /* PAYMENT or ERROR flow */
ikp_creds_t                      *b_creds,       /* buyer's credentials */
ikp_data_t                       *opt_sig_B,     /* opt. for SIG_B */
ikp_error_code_t                 *minor_rc
                                  );

ikp_error_code_t  ikp_Submit_auth_req(
ikp_seller_ctx_t                 *trans_ctx,     /* seller's context */
ikp_acct_priv_creds_t            *my_acct_creds, /* account info for ctx */
ikp_data_t                       *opt_sig_S,     /* optional data for SIG_S*/
ikp_data_t                       *output_token,  /* AUTH_REQ flow */
ikp_error_code_t                 *minor_rc
                                  );

ikp_error_code_t  ikp_Process_auth_resp (
ikp_seller_ctx_t                   *trans_ctx,
ikp_data_t                         *input_token,  /* AUTH_REP flow */
ikp_creds_t                        *a_creds,      /* acquirer's credentials */
```

```
ikp_data_t                         *opt_sig_A,
ikp_data_t                         *output_token,
ikp_error_code_t                   *minor_rc
                                       );

ikp_error_code_t ikp_Submit_clrn_req    (
ikp_seller_ctx_t                    *ctx,
ikp_acct_priv_creds_t               *acct_creds,    /* account info for ctx */
ikp_price_t                         *clrn_amt,      /* amount to be cleared */
ikp_data_t                          *opt_sig_S,     /* opt. data for SIG_S */
ikp_data_t                          *output_token,  /* CLRN_REQ */
ikp_error_code_t                    *minor_rc
                                       );

ikp_error_code_t  ikp_Process_clrn_resp   (
ikp_seller_ctx_t                   *trans_ctx,
ikp_data_t                         *input_token, /* CLEAR_REP flow */
ikp_creds_t                        *a_creds,      /* acquirer's credentials */
ikp_data_t                         *opt_sig_A,
ikp_error_code_t                   *minor_rc
                                      );

ikp_error_code_t  ikp_Compose_cancel   (
ikp_seller_ctx_t                   *ctx,
ikp_acct_priv_creds_t              *acct_creds,    /* in case we need SIG_S */
ikp_respcode_t                     respcode,
ikp_data_t                         *opt_sig_S,    /* optional data for SIG_S*/
ikp_data_t                         *output_token,  /* CANCEL flow   */
ikp_error_code_t                   *minor_rc
                                      );

ikp_error_code_t ikp_Process_inquiry    (
ikp_seller_ctx_t                 *ctx,
ikp_data_t                       *input_token,   /* INQUIRY flow */
ikp_data_t                       *output_token,  /* STATUS flow */
ikp_error_code_t                 *minor_rc
                                       );
```

A.6 Acquirer Functions

This section lists API functions callable on the acquirer side. Invocation of these functions by buyers and sellers causes an error – *WrongRole*; see above.

```
ikp_error_code_t ikp_Process_auth_req  (
ikp_data_t                       *TID_B,          /* buyer's TID */
ikp_data_t                       *TID_S,          /* seller's TID */
ikp_acquirer_ctx_t               **trans_ctx_p,
ikp_data_t                       *input_token,    /* PAYMENT flow */
ikp_data_t                       *opt_sig_S,      /* optional data for SIG_S */
ikp_data_t                       *opt_sig_B,      /* optional data for SIG_B */
ikp_data_t                       *output_token,   /* may return ERROR */
ikp_error_code_t                 *minor_rc
);
```

```
ikp_error_code_t ikp_Process_auth_req1 (
ikp_acquirer_ctx_t                *ctx,
ikp_creds_t                       *b_creds,       /* buyer's credentials */
ikp_creds_t                       *s_creds,       /* seller's credentials */
ikp_acct_priv_creds_t             *a_creds,       /* acquirer's credentials */
ikp_data_t   *output_token,  /* may return ERROR */
ikp_slip_t   *slip,          /* decrypted slip */
ikp_error_code_t   *minor_rc
);

ikp_error_code_t ikp_Compose_auth_resp (
ikp_acquirer_ctx_t   *ctx,
ikp_acct_priv_creds_t   *a_creds,       /* acquirer's credentials */
ikp_data_t   *opt_sig_A,
ikp_respcode_t   respcode,
ikp_data_t   *output_token,  /* may return ERROR */
ikp_error_code_t   *minor_rc
);

ikp_error_code_t ikp_Process_clrn_req   (
ikp_acquirer_ctx_t                   *ctx,
ikp_creds_t                          *s_creds,      /* seller's credentials */
ikp_data_t                           *opt_sig_S,    /* opt. data for SIG_S */
ikp_data_t                           *input_token,  /* CLRN_REQ flow */
ikp_error_code_t                     *minor_rc
                                        );

ikp_error_code_t ikp_Compose_clrn_resp  (
ikp_acquirer_ctx_t                   *ctx,
ikp_acct_priv_creds_t                *a_creds,       /* acquirer's creds */
ikp_data_t                           *opt_sig_A,
ikp_respcode_t                       respcode,
ikp_data_t                           *output_token,
ikp_error_code_t                     *minor_rc
                                        );
```

B iKP State Diagrams

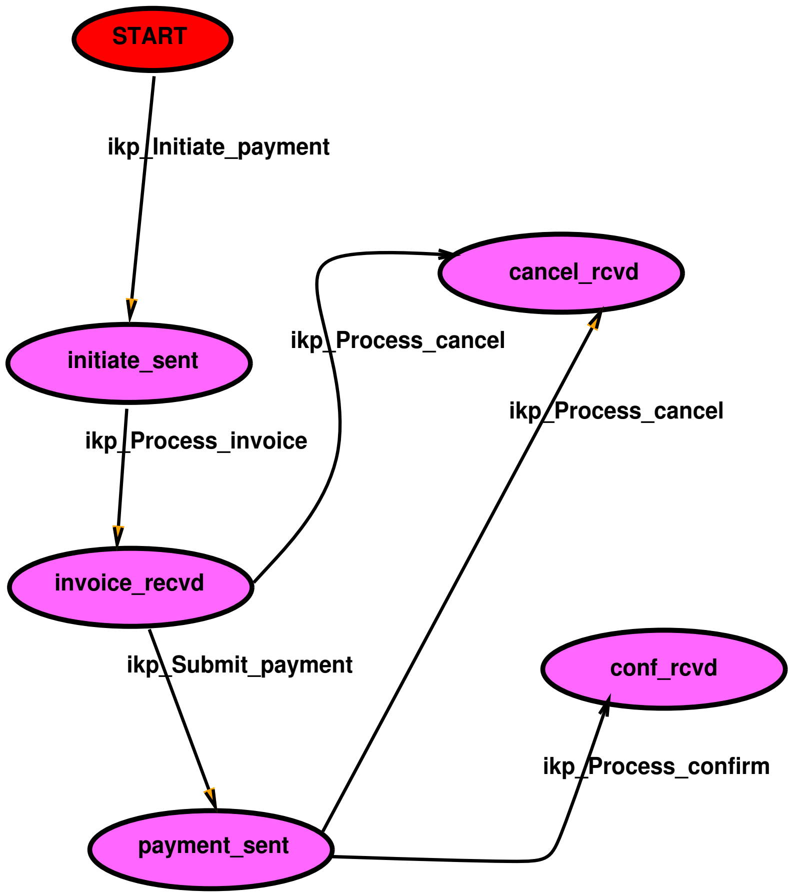

Figure 1: iKP Buyer State Diagram

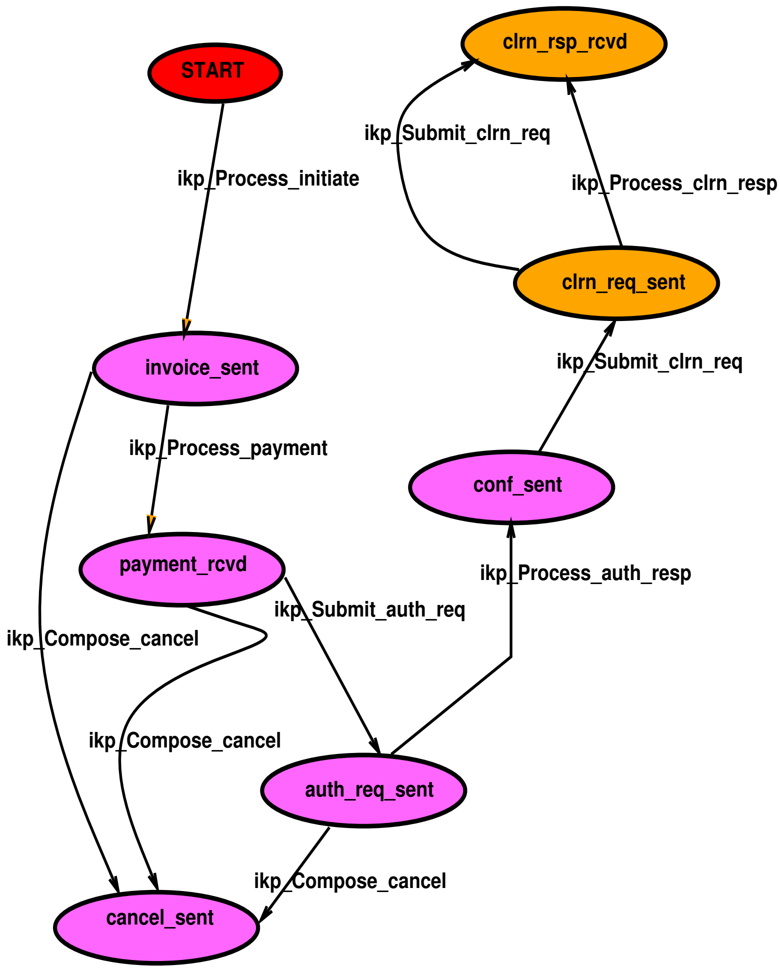

Figure 2: iKP Seller State Diagram

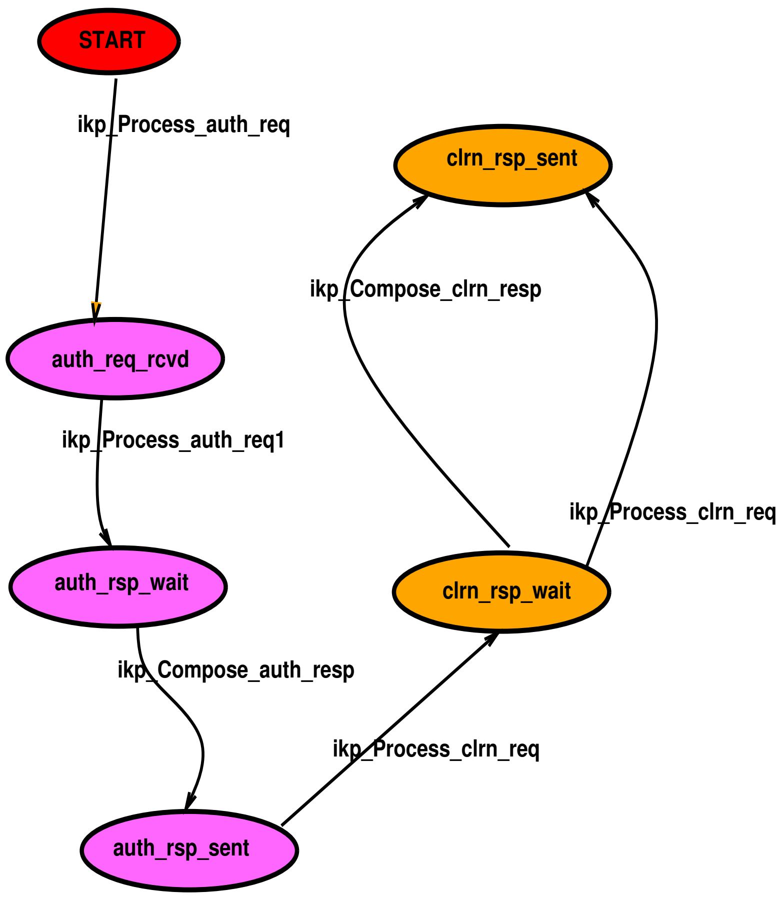

Figure 3: iKP Acquirer Gateway State Diagram